SIAG/OPT Views-and-News

A Forum for the SIAM Activity Group on Optimization

Volume 21 Number 1 October 2010

Contents

Code Generation

Code generation plays an important role in computational science and engineering, and has been used successfully in optimization to compute derivatives from automatic differentiation. The present paper describes a system for code generation that can be used in dynamical parameter estimation problems. The python scripts require text input that describe the model in a few hundred lines of code which are transformed into several hundred thousand of lines of `C++` code that can then be used by nonlinear optimizers such as IPOPT to optimally estimate parameters.

Python Scripting for Dynamical Parameter Estimation in IPOPT

Bryan A. Toth
Department of Physics
Center for Theoretical Biological Physics
University of California, San Diego, La Jolla, CA 92093-0402, USA (`btoth@physics.ucsd.edu`).

1. Introduction

Dynamical state and parameter estimation (DSPE) is an optimization technique where state and parameter estimation, observer theory, and synchronization come together. [1, 11]. This method gives a snapshot of the properties of a nonlinear dynamical system using the minimum amount of data necessary to differentiate the model. In addition to parameter estimation, DSPE estimates the unmeasured state variables of a system, using techniques from dynamical control theory.

In order to implement this method, we use the Python programming language to develop scripts that generate C^{++} files to set-up the problem in the correct format for use with available optimization software. These scripts read two text files which define the vector field of the dynamical system, as well as the feasibility space of the optimization. Using a time-series observation of one (or more) of the state variables, the generated C^{++} code determines the unmeasured parameters of the system, $\mathbf{q}$, as well as the unobserved state variables.

2. Problem Statement

2.1 General Approach

The general problem to be solved is to take a set of first-order differential equations in the state vector

$\mathbf{x}(t) = [x_1(t), \mathbf{x}_\perp(t)]$:

$$\begin{aligned} \frac{dx_1(t)}{dt} &= G_1(x_1(t), \mathbf{x}_\perp(t), \mathbf{q}) \\ \frac{d\mathbf{x}_\perp(t)}{dt} &= \mathbf{G}_\perp(x_1(t), \mathbf{x}_\perp(t), \mathbf{q}). \end{aligned}$$

These equations typically describe an experimental system for which only one of the state variables, $x_1(t)$ can be measured. From this measurement, the unobserved state variables, $\mathbf{x}_\perp(t)$, and the unknown parameters $\mathbf{q}$ are to be determined. If L state variables can be observed, $\mathbf{x}(t) = [x_1(t), x_2(t), \ldots, x_L(t), \mathbf{x}_\perp(t)]$

A typical solution to this type of problem is a least-squares optimization of the error between the measured data, $x_1(t)$ and a model $y_1(t)$; this method works well for linear systems, but breaks down for nonlinear systems with positive conditional Lyapunov exponents (CLEs)[3].

To resolve this problem, the experimental data is coupled to a model system, $\mathbf{y}(t)$ with parameters $\mathbf{p}$, as if for an optimal-tracking problem. This coupling drives the model system to synchronize with the data, and reduces the CLEs to non-positive values. The model functions, $\mathbf{F}$, are chosen to give as close a description as possible to the experimental system; when the experimental system is known precisely, then $\mathbf{F} = \mathbf{G}$.

$$\begin{aligned} \frac{dy_1(t)}{dt} &= F_1(y_1(t), \mathbf{y}_\perp(t), \mathbf{p}) + u(t)(x_1(t) - y_1(t)) \\ \frac{d\mathbf{y}_\perp(t)}{dt} &= \mathbf{F}_\perp(y_1(t), \mathbf{y}_\perp(t), \mathbf{p}). \end{aligned}$$

A term similar to that for $y_1(t)$ is used for each state variable that requires a coupling in order to reduce the CLEs to non-positive values. Again, least-squares optimization could be used on the error between $x_1(t)$ and $y_1(t)$ over the time series of the measured data, but the addition of the coupling term, $u(t)$, complicates matters. This coupling must be chosen large enough to cause synchronization of the data to the model (and eliminate the positive CLEs), but must not overwhelm the underlying dynamics of the system. The addition of the coupling term into the cost function for the optimization will ensure that the coupling does not become too large, while appropriate bounds for the range of the coupling ensure that it becomes large enough. Therefore, the optimization to be performed is (DSPE):

Minimize:

$$C(\mathbf{y}, u, \mathbf{p}) = \frac{1}{2T} \int_0^T \left\{ \Big(x_1(t) - y_1(t) \Big)^2 + u(t)^2 \right\} dt$$

Subject to:

$$\begin{aligned} \frac{dy_1(t)}{dt} &= F_1(y_1(t), \mathbf{y}_\perp(t), \mathbf{p}) + u(t)(x_1(t) - y_1(t)) \\ \frac{d\mathbf{y}_\perp(t)}{dt} &= \mathbf{F}_\perp(y_1(t), \mathbf{y}_\perp(t), \mathbf{p}). \end{aligned}$$

and also subject to suitable bounds for the state variables and parameters.

2.2 Example: Colpitts Oscillator

The Colpitts oscillator is a simple example system to illustrate the implementation of the dynamical state and parameter estimation method. The oscillator has a fairly simple circuit, yet is chaotic over a wide range of parameter values. Dynamical parameter estimation results have previously been reported with this oscillator[3], which we briefly summarize.

The Colpitts oscillator can be described by the following set of differential equations:

$$\begin{aligned} \dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\gamma(x_1 + x_3) - q x_2 \\ \dot{x}_3 &= \eta(x_2 + 1 - e^{-x_1}). \end{aligned}$$

Here, γ, q, and η are unknown parameters. The goal is to determine these unknown parameters, as well as y_2 and y_3 from a time series measurement of x_1, taken over some interval of interest. To differentiate between this data system, and a model system describing it, the data will continue to be described by the $\mathbf{x}$ variable and the model system will be described by $\mathbf{y}$.

The first step is to transform the differential equations into a discrete time map over the interval of interest [0, T]. The choice of numerical integration technique for the differential equations is not unique; we choose Simpson's Rule, with functional midpoints estimated by Hermitian cubic interpolation. This choice gives accuracy of order δt^4; methods with similar accuracy will suffice just as well, but lower order methods must be used carefully to ensure that the data sampling rate, $(\frac{1}{\tau})$, is both accurate enough and

fast enough to capture the underlying dynamics of the system. These integrated equations become the equality constraints in our optimization:

Simpson's Integration:

$$y_i(n+1) = y_i(n) + \frac{\tau}{6}[F_i(n) + 4F_i(n2) + F_i(n+1)]$$

Polynomial Interpolation:

$$y_i(n2) = \frac{1}{2}[y_i(n) + y_i(n+1)] + \frac{\tau}{8}[F_i(n) - F_i(n+1)].$$

For instance,

$$\begin{aligned} & y_1(n+1) \\ &= y_1(n) + \frac{\tau}{6}[y_2(n) + 4\, y_2(n2) + \ y_2(n+1)] \\ & y_1(n2) \\ &= \frac{1}{2}[y_1(n) + y_1(n+1)] + \frac{\tau}{8}[\, y_2(n) - \ y_2(n+1)], \end{aligned}$$

where the $n2$ index refers to the time midpoint between time n and time $n+1$.

The next step is to couple the time series measurement of $x_1(t)$ to its corresponding model variable, $y_1(t)$. This is done in the differential equation by the additional term, $u(t)(x_1(t) - y_1(t))$, so that the discretized integration constraint at each time step for y_1 is:

$$\begin{aligned} y_1(n+1) = & \\ & y_1(n) + \frac{\tau}{6}\Big\{ y_2(n) + u(n)\big(x_1(n) - y_1(n)\big) + \\ & 4\Big(y_2(n2) + u(n2)\big(x_1(n2) - y_1(n2)\big)\Big) + \\ & y_2(n+1) + u(n+1)\big(x_1(n+1) - y_1(n+1)\big)\Big\} \\ y_1(n2) = & \frac{1}{2}[y_1(n) + y_1(n+1)] \\ & + \frac{\tau}{8}\Big\{ y_2(n) + u(n)\big(x_1(n) - y_1(n)\big) \\ & - y_2(n+1) + u(n+1)\big(x_1(n+1) - y_1(n+1)\big)\Big\} \end{aligned}$$

Similar equations are constructed for y_2 and y_3, so that a time series with T data points gives 6T constraint equations for the dynamics of the Colpitts model. The unknown variables are each of the state variables and the coupling at each time step (i.e., $y_1(n)$, $y_2(n)$, $y_3(n)$, $u(n)$), each state variable and coupling at each mid-point time step, and the three parameters (γ, q, and η). In total, there are 4(T+1) + 4T + 4 = 8T+8 unknown variables.

In discretized form, the cost function takes the form of a sum, so the optimization problem is:

Minimize:

$$C(\mathbf{y}, u, \mathbf{p}) = \frac{1}{2T}\sum_{t=0}^{T}\left\{ \Big(x_1(t) - y_1(t)\Big)^2 + u(t)^2 \right\}$$

subject to the 6T constraints above, with appropriate upper and lower bounds for the 8T+8 unknown variables.

A variety of optimization software and algorithms are available to solve this problem. SNOPT[6] and IPOPT[5] were chosen since these are both widely available, are designed for nonlinear problems with sparse Jacobian structure, and can handle large problems. Depending on the problem and the data set, a few thousand data points (or more) are necessary to explore the state space of the model and allow DSPE to produce accurate solutions. For problems of the size of the Colpitts oscillator, this results in tens of thousands of constraints and unknown variables, which can be cumbersome for some solvers. Due to the discretized structure of the problem, however, the Jacobian of the constraints is sparse for these problems, and both SNOPT and IPOPT can take advantage of this sparsity.

2.3 Colpitts Results

The purpose of DSPE is to find parameters and states from experimental systems. In order to test any method for this, a twin experiment is first performed; instead of experimental data, twin data is numerically generated for a known set of parameters and initial conditions. Because the "unknown" parameters are actually known in this scenario, the twin experiment gives a clear indication of the viability of the method for a given system. For the Colpitts oscillator, a twin experiment with 5000 data points was run. The x_1-variable "data" was coupled into the experimental system, and the 3 unknown parameters, as well as the two unmeasured state variables were calculated exactly, as shown in Table 1 and Figures 1 and 2.

An important check is to ensure that the coupling u(t) terms become small as a result of the optimization. Since the synchronization term, $u(t)(x_1(t) - y_1(t))$, is not part of the real dynamical system, this term should be minimized in the dynamics. This is confirmed with the introduction of the 'R-value', which measures the relative contributions of the equation dynamics, $F_1(y_1(t), \mathbf{y}_\perp(t), \mathbf{p})$, and the synchronization term, $u(t)(x_1(t) - y_1(t))$. Formally, the R-value measure is defined as the ratio:

$$\text{R-value} = \frac{[F_1(y_1(t), \mathbf{y}_\perp(t), \mathbf{p})]^2}{[F_1(y_1(t), \mathbf{y}_\perp(t), \mathbf{p})]^2 + [u(t)(x_1(t) - y_1(t))]^2}$$

An R-value is calculated for each equation with a synchronization term. An R-value of 1 at every time point indicates that the optimization found a solution with minimal coupling, while an R-value which varies significantly from 1 indicates that a suitable optimization fit was not made between the data in the model, which may be an indication that the model incorrectly describes the experimental data. For the Colpitts twin experiment, only one R-value is necessary as only one state variable is coupled; in this instance it was calculated to be 1.00 at all time points as expected.

Parameter	Data	Result
γ	0.016	0.016
$\mathbf{q}$	0.14	0.13999
η	1.26	1.2599

Table 1: Colpitts Model: Data Parameters and Estimated Parameters.

After the successful conclusion of the twin experiment, experimental data from a chaotic Colpitts electronic circuit is then used to estimate actual circuit parameters and states, as reported in [3].

This type of twin experiment has been successfully implemented on a wide range of dynamical systems, using nonlinear circuits as described here, spiking neuron models (e.g., Morris-Lecar[9], Hogdkin-Huxley[10]), and simple geophysical fluid flow models, showing the robustness of the dynamic parameter estimation technique.[8]

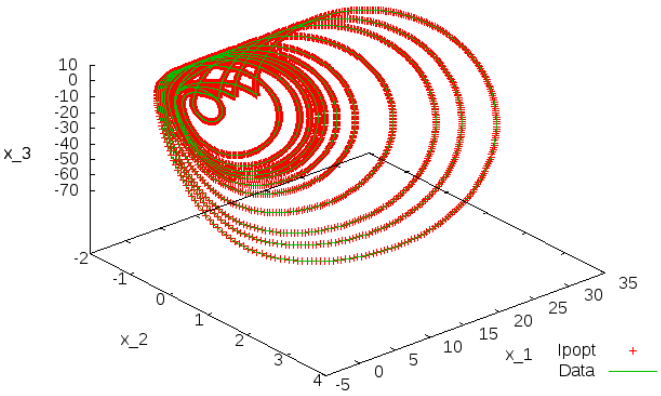

Figure 1: 3-D view of dynamical variable results for the Colpitts oscillator dynamical parameter estimation problem

3. Python Scripting

The basic structure of a dynamic parameter estimation problem is similar for all vector fields - the Colpitts example above was one of the toy models used as a proof of concept for the method. The main goal of this method of parameter and state estimation is to determine unmeasured parameters and states of from real, physical systems.

Each new problem has a unique model associated with it, so a new optimization instance must be constructed. For toy models such as the Colpitts oscillator, the vector field and Jacobian matrix can be readily calculated and input by hand, but this quickly becomes cumbersome for complex models of real systems. To facilitate the use of the dynamical parameter estimation method to a new problem, we have used the Python programming language to develop scripts that set up the problem in the correct format for use with readily available optimization software. These scripts take a simple text file formulation of a parameter estimation vector field and output correctly formatted and linked C^{++} files for use with the widely available IPOPT software libraries. An extension of the scripts to use the SNOPT software libraries is currently under development.

Python is a multi-purpose programming language that permits both object-oriented programming and structured programming. Python can be used as

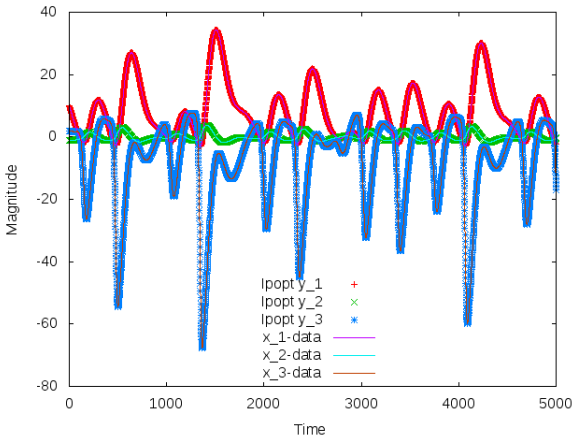

Figure 2: Dynamical variable results for the Colpitts oscillator dynamical parameter estimation problem

a scripting language or a full-fledged programming language, and running an IPOPT or SNOPT optimization can be done with appropriate use of Python modules that interact with C^{++} and Fortran libraries. Python is a higher-level language than C^{++} or Fortran; as a result program development is generally easier, but the execution speed is slower. For the dynamic parameter estimation optimization problems, which currently can consist of tens of thousands of variables and constraints, with the potential for many more, the program execution speed in C^{++} (native language of IPOPT) or Fortran (native language of SNOPT) can be significantly faster than in Python.

The Python scripts define a problem with two distinct text files. The first text file, called 'equations.txt', defines the vector field of the model. The number of equations, parameters, and controls are specified, along with the variable names, model vector field and objective function. For the Colpitts oscillator example, there are three dynamical variables which correspond with the three differential equations of the model, three parameters to be determined, and one control variable. Once this text file is fully defined, the Python module, 'makecode.py' sets up the C^{++} files necessary to run the optimization. The 'equations.txt' file for the Colpitts oscillator dynamic parameter estimation problem is shown in Figure 3.

A second text file, called 'specs.txt' includes problem parameters that the compiled C^{++} executable program loads to run a particular instance of a problem. 'specs.txt' includes the size of the data file (number of data points), sampling frequency of the data, names of the data files needed by the executable, as well as the variable bounds and initial conditions (guess) for the optimization. The 'specs.txt' file for the Colpitts oscillator dynamic parameter estimation problem is shown in Figure 4.

```
# Data file for discretize.py
# Discretize.py skips over any line that begin with #
# Proper format is:
# First line is problem name
# Second line is number of equations, number of parameters,
# number of controls, number of external stimuli
# Following lines list the equations
# With the cost function last
# Then the variable names (in same order as equations)
# Parameter names used in equations
# Control names used in equations
# Data names used in equations
# Problem Name
Colpitts
# nY,nP,nU,nI
3,3,1,0
# equations
yy+k1*(Data-xx)
-gam*(xx+zz)-qq*yy
eta*(yy+1-exp(-xx))
# Objective/Cost function
(Data-xx)*(Data-xx)+k1*k1
# variable names
xx
yy
zz
# parameter names
gam
qq
eta
# control names
k1
# data names
Data
# stimuli names
```

Figure 3: Sample equations.txt file for the Colpitts oscillator dynamical parameter estimation problem

3.1 Discretize.py

The module 'makecode.py' consists of several separate Python scripts, that run sequentially. One of the strengths of the Python language relative to C^{++} and Fortran is the ease of string manipulations. The module that uses this string functionality to set up the optimization problem is 'discretize.py'. 'discretize.py' uses the SymPy package, a Python library for symbolic mathematics[7] for this purpose.

```
# Data file for makecode.py
# Includes the problem length
100
# How much data to skip
# from beginning of data file
1000
# Time step - twice the time step of the data,
# since the data includes time and midpoints.
0.2
# File name - input
colscale1x.dat
# Data File name - stimuli
# No stimuli for this problem
# Boundary & initial conditions
# 0 for no initial data file, 1 for data file
# A data file must include values for all state
# variables at each time point.
0
# If above is 1, list name of data file next.
# If 0, no entry needed.
# State Variables:
# These are in the formats:
# lower bound, upper bound, initial guess
# Boundary & initial conditions
# x
-100, 100, 0.0
# y
-100, 100, 10
# z
-100, 100, 0
# k
0, 100, 0
# dk
# derivative of the control parameter
-1, 1, 0
# p1
0, 100, 5
# p2
0, 100, 10
# p3
0, 100, 20
```

Figure 4: Sample specs.txt file for the Colpitts oscillator dynamical parameter estimation problem

The state variable, parameter, and control names, as well as the vector field and objective function for the problem are imported from 'equations.txt' and converted into symbolic equations for use in SymPy. These symbolic equations are then transformed into a discretized integration according to Simpson's Rule with polynomial interpolation described above.

This discretized form of the vector field is then used to symbolically calculate the Jacobian and Hessian matrices, using SymPy to take first and second derivatives of the vector field with respect to all the state variables, parameters, and controls. For the Simpson's Rule discretization choice, care is taken to keep track of whether state variable and control derivatives are taken with respect to the variable at the current time, next time, or mid-point time, to ensure proper placement within the Jacobian and Hessian structures. The result is symbolic arrays for the vector field, Jacobian, and Hessian, that include only non-zero entries only, since these matrix elements will be entered into the optimization in a standard sparse matrix formulation that only needs the row, column, and value information for non-zero entries. Five separate arrays from 'discretize.py' are needed in the IPOPT optimization:

- Objective function
- Constraints
- Gradient of the Objective Function
- Jacobian of the Constraints
- Hessian of the Objective Function and Constraints

The symbolic strings in these arrays are converted into proper C^{++} format (e.g., ** changed to pow function) and stored for the next part of the process, 'makecode.py' itself.

3.2 Makecode.py

A typical C^{++} IPOPT optimization program consists of a main program that initiates a new problem class and calls the optimization process. The problem class is typically defined in another file, with an appropriate header file. For our purposes, the main program is standard across different vector fields; the details of the vector fields are contained within the other files, which we call problemname_nlp.cpp and subsequent header file, problemname_nlp.hpp. The module 'makecode.py' takes the information outputted from 'discretize.py' and writes these C^{++} files for a particular problem. The executable C^{++} program, once compiled, loads the information in specs.txt in order to run the program. In this way, a given vector field can be sampled over various data sets of differing lengths, without re-compilation of the program.

An example of one of the routines that 'makecode.py' generates is the 'Eval-g' function, which returns the value of the constraints. These constraints have been stored in an array in symbolic discretized form by 'discretize.py'. The code generation for this

function consists of setting up a loop over all time points, so that each discretized constraint is defined at each time point. Each symbolic variable from 'discretize.py' is equated to the proper term of a C^{++} array that contains all optimization variables. The 'Eval-g' function for the Colpitts oscillator problem defined by the Figure 3 'equations.txt' file is shown as Figure 5. This function is rather straightforward; for the three dynamical variable oscillator, 42 lines of C^{++} code is generated to define these as constraints. The functions that define the Jacobian and Hessian of the vector field are significantly more complicated, since both functions require detailed row and column information that defines the constraint and partial derivative that is being taken. The total length of generated C^{++} code for each function for the Colpitts oscillator is shown in Table 2.

Routine	Description	Lines
Eval_f	Evaluate Objective	68
Eval_grad_f	Evaluate Objective Gradient	70
Eval_g	Evaluate Constraints	42
Eval_jac_g	Evaluate Jacobian	218
Eval_h	Evaluate Hessian	209

Table 2: Length of generated code for various subroutines in Colpitts example.

Several other Python modules are used to produce the necessary C^{++} code, as well as a basic Makefile to link the files together at compilation. The general structure of these files is similar across different problems, with the main difference being the vector field, Jacobian matrix, and Hessian matrix. For large problems, defined in terms of the number of dynamical variables (or number of equations), the generated C^{++} code can be very large, as shown in Table 3.

Model	Type	Vars	Cons	equations.txt	C^{++} lines
Colpitts	Osc	10T+8	7T	37 lines	996
Lorenz	Atm	18T+10	12T	39 lines	1394
HH	Neur	12T+28	9T	57 lines	2171
lobster	Neur	44T+86	35T	145 lines	6258
Arakawa	Ocean	384T+195	192T	283 lines	218034

Table 3: Length of generated code for various dynamical parameter estimation problems. The listed problems are the Colpitts oscillator discussed herein, the Lorenz atmospheric model[12], the Hodgkin-Huxley neuron model[10], the lobster lateral pyloric neuron model[14], and the barotropic vorticity ocean circulation model[13]

4. Discussion

Dynamical parameter estimation has applications in a wide range of fields, and these Python scripts have made the implementation for a new model to be straightforward. The version discussed here uses Simpson's integration rule and the IPOPT solver, but these can be easily substituted. For instance, the integration rule is defined in just a few lines of the discretize.py code, and can be changed out to another rule fairly simply. Use of another optimization solver is slightly more complex since program syntax varies across solvers, but the general front-end algorithms are similar among optimization software that makes use of the standard sparse matrix formulation that only needs the row, column, and value information for non-zero entries.

More importantly, no language-specific programming knowledge is necessary in order to use these scripts for dynamical parameter estimation. The scripts are written in Python, but are run from the a terminal command line, so no Python-specific knowledge is needed. Optimization software packages typically include interfaces to allow the use of a user's language of choice, but these interfaces may not be as well-supported as the base software, may be inefficient to use, and may significantly slow down the computational time to solve a given problem. By writing code in the native language of the optimization software, these Python scripts require only basic command line skills to solve complicated dynamical parameter estimation problems.

Python is the natural programming language to implement this type of program due to its string handling ability. Higher level programs such as MATLAB and Mathematica are capable of implementing the symbolic differentiation performed by SymPy, and many programming languages can be set to write text to a file (which is what the Python scripts ultimately do). Many programming languages struggle to seamlessly import information from a text file, and this is where Python excels. These Python scripts would not be necessary if only a single optimization instance needed to be performed on a unique model vector field. The ability to radically

change the system being studied by editing just two text files opens the door for dynamical parameter estimation to be easily implemented across many fields.

Acknowledgments

This work was supported by the Center for Theoretical Biological Physics (NSF-PHY-0822283).

REFERENCES

[1] Creveling, D., P. E. Gill, and H. D. I. Abarbanel, "State and Parameter Estimation in Nonlinear Systems as an Optimal Tracking Problem," *Physics Letters A*, **372**, 2640-2644 (2008).

[2] Pecora, L. M. and T. L. Carroll, "Synchronization in chaotic systems," *Physical Review Letters* **64**, 821–824 (1990).

[3] Abarbanel, H. D. I., D. R. Creveling, R. Farsian, and M. Kostuk, "Dynamical State and Parameter Estimation," *SIAM Journal of Applied Dynamical Systems* **8**, 1341-1381 (2009).

[4] Fourer, R., D. M. Gay, and B. W. Kernighan, "A Modeling Language for Mathematical Programming," *Management Science* **36**, 519-554 (1990).

[5] *W*ächter, A., and L. T. Biegler, "On the Implementation of a Primal-Dual Interior Point Filter Line Search Algorithm for Large-Scale Nonlinear Programming," *Mathematical Programming* **106(1)**, 25-57 (2006).

[6] Gill, P.E., W. Murray, and M.A. Saunders, "SNOPT: An SQP Algorithm for Large-scale Constrained Optimization," *SIAM Review* **47(1)**, 99-131 (2005).

[7] Certik, O., "SymPy Library for Symbolic Mathematics," *Technical report, http://code.google.com/p/SymPy/*, since 2006.

[8] Abarbanel, H. D. I., P. Bryant, P. E. Gill, M. Kostuk, J. Rofeh, Z. Singer, B. Toth, and E. Wong, "Dynamical Parameter and State Estimation in Neuron Models", *The Dynamic Brain: An Exploration of Neuronal Variability and Its Functional Significance*, eds. D. Glanzman and Mingzhou Ding, Oxford University Press (2010).

[9] Morris, C. and M. Lecar, "Voltage oscillations in the barnacle giant muscle", *Biophys. J.* **71**, 3030–3045 (1981).

[10] Hodgkin, A. L., and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve." *J. Physiol.*, 10:500-544 (1952).

[11] Nijmeijer, H., "A dynamical control view on synchronization." *Physica D***154**, 219-228 (2001).

[12] Lorenz, E. N., "Predictability - A problem partly solved." *Proceedings of the Seminar on Predictability*, volume 1. ECMWF: Reading, UK, 1-18 (1996).

[13] Arakawa, A., "Computational design for long-term numerical integration of the equations of fluid motion: Two-dimensional incompressible flow." *J. of Comp. Physics* **1**, 119-143 (1966).

[14] Nowotny, T., R. Levi, and A. I. Selverston, "Probing the Dynamics of Identified Neurons with a Data-Driven Modeling Approach." *PloS ONE* 3:e2627 (2008).

```
bool COLPITTS_NLP::eval_g(Index n, const Number* x,
                  bool new_x, Index m, Number* g)
{
assert(n == 10*Time+8);
assert(m == 7*Time);
for(Index jt=0;jt<Time;jt++) {
  for(Index i=0;i<nY;i++) {
    Xval[i] = x[jt + i*(Time+1)];
    Xvalp1[i] = x[jt + i*(Time+1) + 1];
    Xval2[i] = x[(Time+1)*(nY+2*nU) + jt + i*(Time)];
  } //end for loop
  for(Index i=0;i<nU;i++) {
    K11val[i] = x[jt + nY*(Time+1) + 2*i*(Time+1)];
    K11valp1[i] = x[jt + nY*(Time+1) + 2*i*(Time+1) + 1];
    K11val2[i] = x[(Time+1)*(nY+2*nU)+(nY+2*i)*Time+jt];
    dK11val[i] = x[jt + (nY+2*i+1)*(Time+1)];
    dK11valp1[i] = x[jt + (nY+2*i+1)*(Time+1)+1];
    dK11val2[i] = x[(Time+1)*(nY+2*nU)+(nY+2*i+1)*Time+jt];
  } //end for loop

  Xdval[0] = Data[2*jt];
  Xdval2[0] = Data[2*jt+1];
  Xdvalp1[0] = Data[2*jt+2];

  for(Index i=0;i<nP;i++) {
    Pval[i] = x[(2*Time+1)*(nY+2*nU)+i];
  } //end for loop

  g[7*jt+0] = Xval[0] + 0.167*hstep*(Xval[1] +
         K11val[0]*(Xdval[0] - Xval[0])) - Xvalp1[0] +
         0.167*hstep*(Xvalp1[1] + K11valp1[0]*
         (Xdvalp1[0] - Xvalp1[0])) + 0.667*hstep*
         (Xval2[1] + K11val2[0]*(Xdval2[0] - Xval2[0]));
  g[7*jt+1] = Xval[1] + 0.167*hstep*(-Pval[0]*
         (Xval[0] + Xval[2]) - Pval[1]*Xval[1]) -
         Xvalp1[1] + 0.167*hstep*(-Pval[0]*
         (Xvalp1[0] + Xvalp1[2]) - Pval[1]*Xvalp1[1]) +
         0.667*hstep*(-Pval[0]*(Xval2[0] + Xval2[2]) -
         Pval[1]*Xval2[1]);
  g[7*jt+2] = Xval[2] + 0.167*Pval[2]*hstep*(1 +
         Xval[1] - exp(-Xval[0])) + -Xvalp1[2] +
         0.167*Pval[2]*hstep*(1 + Xvalp1[1] -
         exp(-Xvalp1[0])) + 0.667*Pval[2]*hstep*
         (1 + Xval2[1] - exp(-Xval2[0]));
  g[7*jt+3] = 0.5*Xval[0] + 0.125*hstep*(Xval[1] +
          K11val[0]*(Xdval[0] - Xval[0])) + 0.5*Xvalp1[0] -
          0.125*hstep*(Xvalp1[1] + K11valp1[0]*
          (Xdvalp1[0] - Xvalp1[0])) + -Xval2[0];
  g[7*jt+4] = 0.5*Xval[1] + 0.125*hstep*(-Pval[0]*
          (Xval[0] + Xval[2]) - Pval[1]*Xval[1]) +
          0.5*Xvalp1[1] - 0.125*hstep*(-Pval[0]*
          (Xvalp1[0] + Xvalp1[2]) -
          Pval[1]*Xvalp1[1]) -Xval2[1];
  g[7*jt+5] = 0.5*Xval[2] + 0.125*Pval[2]*hstep*
           (1 + Xval[1] - exp(-Xval[0])) +
           0.5*Xvalp1[2] - 0.125*Pval[2]*hstep*
           (1 + Xvalp1[1] - exp(-Xvalp1[0])) -Xval2[2];
  g[7*jt+6] = 0.5*K11val[0] + 0.125*dK11val[0]*hstep +
           0.5*K11valp1[0] - 0.125*dK11valp1[0]*hstep
           -K11val2[0];

} //end for loop

return true;
}
```

Figure 5: Sample eval_g C^{++} routine for the Colpitts example

Bulletin

IFIP TC7 Conference

Call for Papers
25th IFIP TC7 Conference on System Modeling and Optimization
September 12-16, 2011, Berlin, Germany
http://www.ifip2011.de

Conference Topics:

- optimization theory
- linear and nonlinear programming
- stability and sensitivity analysis
- stochastic optimization
- combinatorial and discrete optimization
- large-scale optimization
- optimal control governed by ODEs and PDEs
- industrial applications of optimization
- modeling and optimization in information processing

Important Deadlines and Information. The deadline for submission of minisymposium proposals is September 1, 2010, the deadline for submission of contributed talks/posters is November 1, 2010. For more information about this conference contact the website http://www.ifip2011.de

Deadlines for SIAM OP11

Submission & Conference Deadlines.

- October 18, 2010: Minisymposium proposals
- November 15, 2010: Abstracts for contributed and minisymposium speakers
- April 4, 2011: Pre-Registration

Travel Fund. SIAM provides travel support for students, postdocs, and early-career scientist. The deadline for applications is November 1, 2010.

Nominations for SIAG/OPT Prize: November 15, 2010.

Call for Manuscripts for the MPS-SIAM Series on Optimization

The joint MPS-SIAM book series seeks high-quality texts from all areas of optimization. We welcome research monographs on cutting-edge topics, books on applications, textbooks at all levels, and tutorials. Rigorous selection guarantees that all books meet the highest quality standards, making this a flagship collection in the field.

The series offers many benefits to its authors and customers:

- Books can be marketed to over 100,000 people via membership and other pertinent mailing lists.
- All books remain in print until they are replaced by an updated edition.
- Royalties are competitive with those of other publishers.
- All books are fully copy edited by experienced staff.
- MPS and SIAM members enjoy a 30% discount off list price, and nonmembers get a 20% discount at conference exhibits.
- Instructors using series books for their courses can get a 20% discount for their students.
- Due to a distribution partnership, customers outside North America can order books through Cambridge University Press using their local currency.
- Publishing with the series supports SIAM's mission to promote, advance, and provide programs and media for the applied mathematics and computational science communities.

If you have a book topic suggestion or are writing a book yourself, please contact series editor-in-chief Tom Liebling (thomas.liebling@epfl.ch) or series acquisitions editor Sara Murphy (murphy@siam.org) for more information.

A list of books in the series can be found at http://www.siam.org/catalog/mp.php.

Chairman's Column

Organization for the 10th SIAM Conference on Optimization is going ahead full steam — that's to both the program committee and the local organizers for their hard work on our behalf. I hope that you are all working on minisymposia submissions — the deadline for this and other things associated with next May's conference are given in the bulletin section above.

I'd like to thank Erling Andersen, Kurt Anstreicher, Martine Labbe, Ariela Sofer and Luis Vicente for serving on the "Officer Nominating Committee" to replace our current officers for the next 3 year cycle. The excellent slate of candidates they have produced can be found on our wiki at: `http://wiki.siam.org/siag-op` along with other information regarding the activities of the SIAG. Please make sure you cast your votes for your favored candidates when the announcement is made via email.

There continues to be a growing collection of blogs and information regarding optimization, its theory, algorithms and applications that becoming available online. I would welcome suggestions (to my email) on how the SIAG can become more a part of this, and to help move this old fogie into the 21st century. Maybe someone can write a summary article on this for the next Views-and-News, and show us how to reach the next generation of researchers and students in our disciplines. Or simply help us keep informed and aware of all the information that we could use...

Michael C. Ferris, SIAG/OPT Chair
Computer Sciences Department
University of Wisconsin-Madison
1210 West Dayton Street,
Madison, WI 53706
USA
`ferris@cs.wisc.edu`
`http://pages.cs.wisc.edu/~ferris/`

Comments from the Editor

Response to Michael's Challenge. At the last SIAM annual meeting in Pittsburgh, SIAM experimented with `twitter`. After a shy start-up period, the participants generated some lively tweets, including the first-ever "tweetable" talk by David Gleich (Stanford). You can relive those fun days by scrolling through the tweets at `http://twitter.com/SIAMconnect` . Maybe SIAG/OPT can get it's own list, or tweet during OP11!

New Submissions for Views. Views-and-News always needs new papers. Please consider submitting a short paper (maybe summarizing a paper that you submitted to another journal), or suggest a topic for the next issue. There are two advantages to publishing in Views-and-News: papers are reviewed lightning fast, and longer versions can later be published in regular journals.

Sven Leyffer, Editor
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439, USA
`leyffer@mcs.anl.gov`
`http://www.mcs.anl.gov/~leyffer/`