$$q = \mathbf{Q}u$$

$$\mathbf{M}u + \mathbf{G}^{\mathsf{T}}\lambda = \mathbf{f} - \mathbf{f}_{\text{bias}}$$

$$\mathbf{G}u + \mathbf{b} = 0$$

Simmath User's

What are “numerical methods”?

SimTK Simmath™ 2.1
User's Guide

Abstract

1 Introduction

1.1 Simmath ancestry

1.2 Document conventions

API: "Application Programming Interface," i.e., a programming library.

typewriter font

SimTK

Optimizer SimTK::Optimizer

using

"using namespace SimTK;" to introduce all SimTK

targeted statements like "using SimTK::Optimizer;" which allows use of only the name "Optimizer" without the prefix.

macro names, we always start the name with the characters "SimTK_", capitalized exactly as

2.4 Scalars, vectors and matrices

SimTK::Vector SimTK::Matrix

Simmatrix User's Guide

Real	double	typedef
Vector	*m*x	Real
RowVector	x*n*	Real

Matrix	mxn Real m n

Vector Matrix

The "~" (tilde) operator is overloaded to indicate the transpose operation, so that for ~m m^T Matrix m Matrix Vector mx RowVector xn Vector RowVector

[] 0

v[i] r[j] i Vector RowVector m[i] i RowVector Matrix ()

Vector RowVector *column* *row* Matrix m(j) j Matrix m Vector m(i,j) i-j m * All indexing operations produce "lvalues,"

3 Linear algebra

You can also select elements with the more "C like" construct m[i][j] m(i,j)

3.1 Solving Linear Systems (SimTK::FactorLU)

3.2 Linear Least Squares (SimTK::FactorQTZ)

3.3 Singular Value Decomposition (SimTK::FactorSVD)

3.4 Eigen Values (SimTK::Eigen)

4 Numerical differentiation (SimTK::Differentiator)

n n

n x n $\mathbf{J} = \partial\mathbf{f}/\partial\mathbf{y}$

0

’s source code to produce source for

. Such methods are called “numerical differentiation” and Simmath p `Differentiator`

Differentiator

Differentiator's abstract function classes. The most general

JacobianFunction

GradientFunction

f ScalarFunction

$f\,y$

```
Differentiator::JacobianFunction
Differentiator::GradientFunction
Differentiator::ScalarFunction
```

Differentiator::Function

4.1 Example

$f\,x$ ωx

ω $f'\,x$ f x

x $f'\,x$ ω ωx

Differentiator

The program's output is shown in bold at the end.

```cpp
#include "simmath/Differentiator.h"
#include <cstdio>
#include <cmath>
#include <exception>
using SimTK::Real;
using SimTK::Differentiator;

// user-written class
class SinOmegaX : public Differentiator::ScalarFunction {
public:
    SinOmegaX(Real omega) : w(omega) { }

    // Must provide this virtual function.
    int f(Real x, Real& fx) const {
        fx = std::sin(w*x);
        return 0; // success
    }
private:
    const Real w;
};dsinwx(sinwx);

int main () {
try
  { const Real w=3;
    SinOmegaX      sinwx(w);
    Differentiator dsinwx(sinwx);

    const Real x = 1.234;
    Real exact  = w*std::cos(w*x);
```

(w*x);

ε Function

4.3 Available algorithms

forward difference *central difference*

$\varepsilon \approx \sqrt{\varepsilon_f}$

$y_i \in \mathbf{y}$

$\varepsilon \approx \varepsilon_f^{\frac{2}{3}}$

ε

much

ε ≈

ε ≈

ε ≈ x

ε ≈ε

5 Time stepping and numerical integration (SimTK::TimeStepper, SimTK::Integrator)

initial conditions

time stepping *Numerical integration*

stepping used to advance through “smooth” intervals of the overall trajectory.

hybrid

systems

time stepper’s numerical integrator is twofold: (1)

events

report time

requested report times; the system's continuity is uninterrupted by reporting.

stepper. The time stepper invokes the hybrid system's *event handler*

integrator's internal context is maintained between calls, however, so when the time stepper

which is achieved by the integrator's internal step s

not

In practice this is not strictly true since an integrator may perform a little "rounding off"

5.1 The continuous system

$$\dot{y} = f(d;t,y)$$
$$0 = c(d;t,y)$$
$$0 = e(d;t,y)$$

t y d

$\dot{y} = dy/dt$

y d

d

$$\text{sign}[e(t, y_t)] = \text{sign}[e_0^I]$$

$t > t_0^I$ x

x

5.2 The discrete system

d

. The integrator's task is to isolate

t

$$\text{sign}[e(t, y_t)] \neq \text{sign}[e(t - \varepsilon, y_{t-\varepsilon})]$$

example, an event may be triggered only upon a "rising" transition (sign[e_i t e_i t–
or "falling" transition (sign[e_i t e_i t–

i

handling

invokes the system's event handler on the state in the condition where the event(s) have

time with the "fixed up" state.

d

r_0 d

`r=dist y` r

$$e_i(y) = \qquad (y, \mathrm{p}_1, \mathrm{p}_2) - r_0$$

r_0

d

d

d hasn't already been set, like this:

$$e_i(y) = \begin{cases} d_{\text{flag}}, & 0 \\ !d_{\text{flag}}, & \texttt{dist}(y, \mathrm{p}_1, \mathrm{p}_2) - r_0 \end{cases}$$

(falling only)

r_0

d

inuous event triggers are allowed but have to be localized by "binary chopping" which

5.2.1 Event localization

t

t

localize t t

event window

t t $\leq t$ *not* t

has t , but we're not sure of the precise value of t

t

acceptable then the integrator's t

Once an event has been localized to an acceptable tolerance, the integrator's `stepTo()`

t

5.2.2 Event handlers

e a state in an “event(s) triggered” condition,

5.2.3 Other event types

- *t* (“scheduled events”)
- “end of step” updates (“time advanced events”)
-
-

`stepTo()`

declare that a scheduled event has occurred, call the system’s event handler, and reinitialize

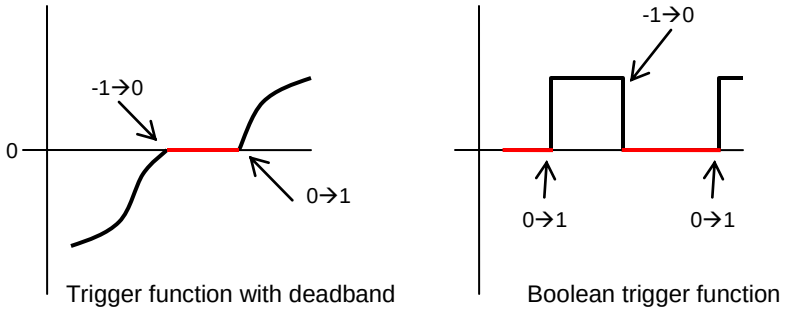
however the hybrid system can request that if necessary, in which case “end of step” is

“final time” for the simulation, or when an event handler indicates that the simulation must

5.2.4 Event trigger transition details

crossings

occurs when the trigger's value goes from negative to positive or vice versa at some moment



-1→1 1→-1

the trigger function has a discrete zero “zone” (or “deadband”) and we will report transitions

prior

-1→1 1→-1

-1→0 1→1, then we’ll

-1→0 -1→1

Significant sign transitions		Transition seen before localization	Localized transition	Reported transition
Rising -1→1	continuous trigger	-1→1	any rising	-1→1
		-1→0	(unchanged)	-1→1
		0→1	*no event*	
	discrete zero adds -1→0, 0→1	-1→1	any rising	report localized transition
		-1→0	(unchanged)	-1→0
		0→1	(unchanged)	0→1
Falling 1→-1	continuous trigger	1→-1	any falling	1→-1
		1→0	(unchanged)	1→-1
		0→-1	*no event*	
	discrete zero adds 1→0, 0→-1	1→-1	any falling	report localized transition
		1→0	(unchanged)	1→0
		0→-1	(unchanged)	0→-1

to *from*

zero event trigger, if the function doesn't stay zero long

→ →

5.3 Accuracy, scaling, tolerances

0 ≤1

"within 0.1%" of the "perfect answer." (These phrases are in quotations because they are

n_d $n_d = -\log_{10}\alpha$

$\alpha = 1\text{e-}5 \Rightarrow n_d = 5$

exactly what they mean by "1% accuracy." Nevertheless we feel it is important to provide a

single “knob” for a user to turn that delivers “more accuracy” at higher cost or “less ccuracy” at lower cost, in a way that at least attempts to capture what at typical user might

The primary difficulty we encounter is that the variables and equations defining the user’s

5.3.2 Weights and constraint tolerances

$w_i \geq 0$ y_i t_i 0

c_i w_i's on its diagonal, and

t_i

reciprocal tolerances "constraint weights"). Then given the fractional accuracy specification

y

ε_y ≤ $c\ t\ y$ ≤

ε_y is the vector of estimated absolute errors in each state variable y, as estimated by the integrator for a trial step under consideration.

5.3.3 Event localization window

e_i a "unit" localization

l_i (in units of the system's time scale

e_i ≤ l_i

t t $t_{\text{high}} - t_{\text{low}} \leq \alpha\tau l_i, \quad \forall i \in \mathrm{E}$

5.3.4 Default accuracy, scaling and tolerances

accuracy	$\alpha = 0.001$
time scale	$\tau = 1$
weights	$w_i = 1, \quad 0 \leq i < n_y$
constraint tolerances	$t_i = 0.1, \quad 0 \leq i < n_c$
localization windows	$l_i = 0.1, \ 0 \leq i < n_e$

"*rtol atol*" scheme is achieved by defining

$$w_i = \frac{1}{\max(|y_i|, u_i)}$$

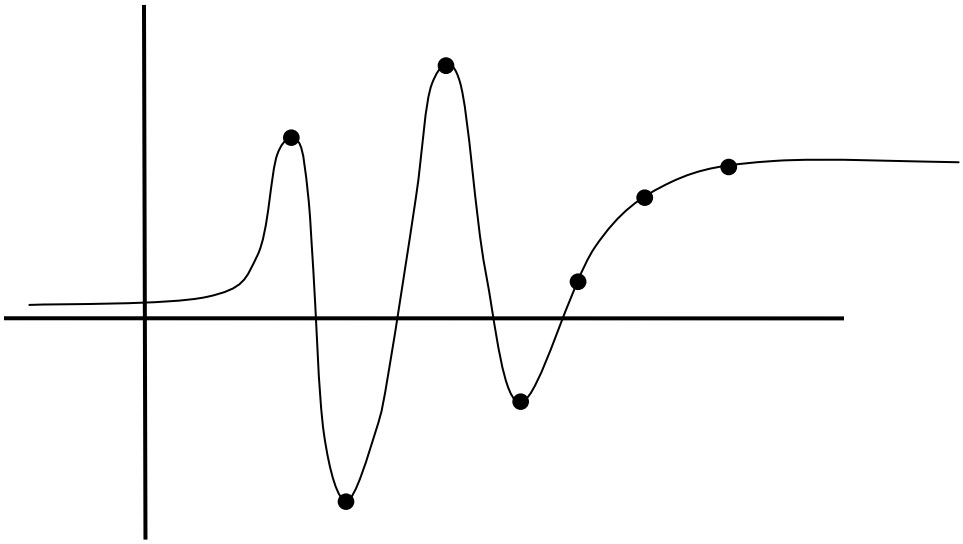
u_i represents "one unit" of error in y_i

rtol *atol*$_i$ u_i

6 Numerical optimization (SimTK::Optimizer)

SimTK::Optimizer

Optimizer

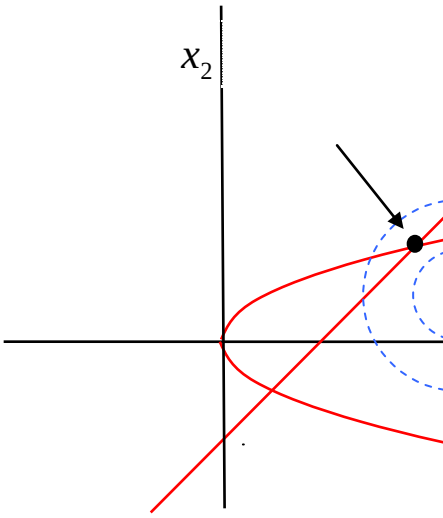


–

$$(x_1-5)^2+(x_2-1)^2$$

$$x_1 - x_2^2 \geq 0$$

$$-x_1 + x_2 + 2 \geq 0$$



Optimizer

OptimizerSystem Optimizer

OptimizerSystem

OptmizerSystem

OptimizerSystem

objectiveFunction()

Optimizer

OptimizerSystem

by calling the Optimizer class's optimize() Optimizer.optimize()

SimTK::Vector

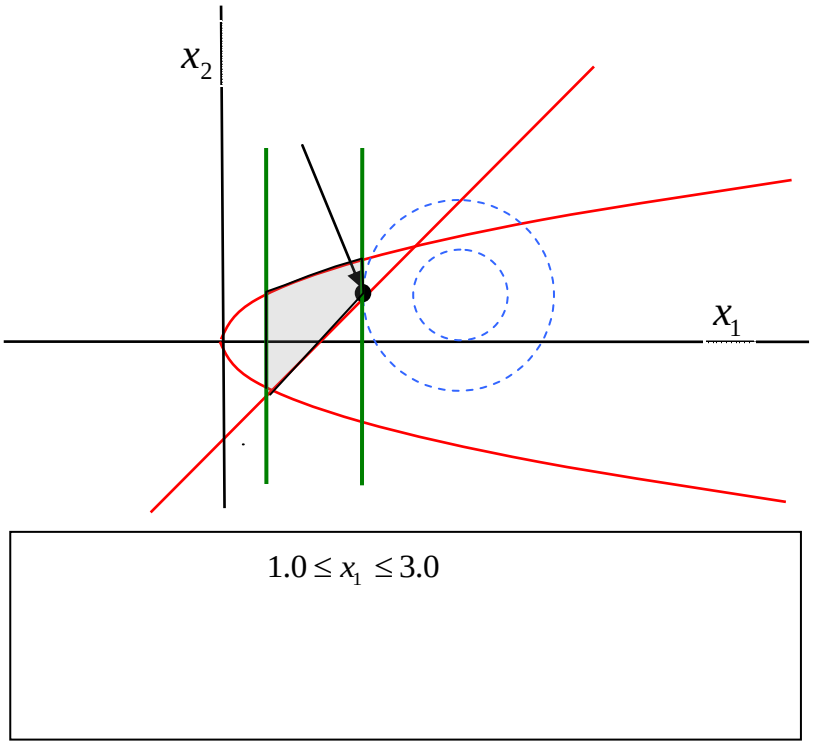
Vector Optimizer.optimize()

Optimizer

```
#include "SimTKmath.h"
#include <iostream>

static int NUMBER_OF_PARAMETERS = 2;
static int NUMBER_OF_EQUALITY_CONSTRAINTS = 0;
static int NUMBER_OF_INEQUALITY_CONSTRAINTS = 2;

// user-written class
class ProblemSystem : public OptimizerSystem {
public:

    // Must provide this virtual function.
   int objectiveFunc(  const Vector &coefficients, const bool new_coefficients,
Real& f ) const {
      const Real *x;
      int i;

      x = &coefficients[0];

      f = (x[0] - 5.0)*(x[0] - 5.0) + (x[1] - 1.0)*(x[1] - 1.0);
      return( 0 );
   }
   int gradientFunc( const Vector &coefficients, const bool new_coefficients,
Vector &gradient ) const{
      const Real *x;

      x = &coefficients[0];

     gradient[0] = 2.0*(x[0] - 5.0);
     gradient[1] = 2.0*(x[1] - 1.0);

     return(0);


  }
  /*
  ** Method to compute the value of the constraints.
  ** Equality constraints are first followed by the any inequality constraints
  */
  int constraintFunc( const Vector &coefficients, const bool new_coefficients,
Vector &constraints)  const{
      const Real *x;

      x = &coefficients[0];
      constraints[0] = x[0] - x[1]*x[1];
      constraints[1] = x[1] - x[0] + 2.0;

      return(0);
  }


  /*
  ** Method to compute the Jacobian of the constraints.
```

{

```
      jac(1,0) = -1.0;
      jac(1,1) =  1.0;


      return(0);
  }


   ProblemSystem( const int numParams, const int numEqualityConstraints,
                  const int numInequalityConstraints) :

         OptimizerSystem( numParams )
   {
         setNumEqualityConstraints( numEqualityConstraints );
         setNumInequalityConstraints( numInequalityConstraints );
   }

};
main() {

    Real f;
    int i;

    /* create the system to be optimized */
    ProblemSystem sys(NUMBER_OF_PARAMETERS, NUMBER_OF_EQAULITY_CONSTRAINTS,
                      NUMBER_OF_INEQUALITY_CONSTRAINTS );

    Vector results(NUMBER_OF_PARAMETERS);


    /* set initial conditions */
    results[0] = 5.0;
    results[1] = 5.0;

   try {

      Optimizer opt( sys );

      opt.setConvergenceTolerance( .0000001 );

      /* compute  optimization */
      f = opt.optimize( results );
   }
   catch (const std::exception& e) {
      std::cout << "ConstrainedOptimization.cpp Caught exception:" << std::endl;
      std::cout << e.what() << std::endl;
   }


    printf("Optimal Solution: f = %f  parameters = %f %f\n", f, results[0],
    results[1]);

    return 0;
}
```



x_1 $1.0 \le x_1 \le 3.0$

```
Vector lower_limits(NUMBER_OF_PARAMETERS);
Vector upper_limits(NUMBER_OF_PARAMETERS);

/* set limits on the parameters */
   lower_limits[0] =  1.0;
   upper_limits[0] =  3.0;
```

```
   lower_limits[1] = -2e19;
   upper_limits[1] =  2e19;

sys.setParameterLimits( lower_limits, upper_limits );
```

7 Other Mathematical Tools

7.1 Random Numbers (SimTK::Random)

lgorithm for generating random numbers is more accurately known as a “pseudo random number generator”

sequence, you can do this by initializing each one with a different “seed” value. Every

`SimTK::Random`

`SimTK::Random`

`SimTK::Random::Uniform` `SimTK::Random::Gaussian`

```
Random::Uniform random(0.0, 100.0);
// Each time you call getValue(), it will return a different value.
Real nextValue = random.getValue();
```

7.2 Roots of Polynomials (SimTK::PolynomialRootFinder)

```
Vec4 coefficients(1.0, -6.0, 11.0, -6.0);
Vec<3,Complex> roots;
PolynomialRootFinder::findRoots(coefficients, roots);
cout << "Roots: " << roots << endl;
```

References