Constructing Scientific Programs with SymPy

Mark Dewing

July 14, 2011
Outline

Motivation and Overview of Writing Scientific Programs

Implementation of a Framework

Example: Partition Function Integral
Writing Scientific Programs by Hand

Derive equations

1. \( \frac{df}{dx} = 2x \)
2. \( \frac{f(x+h) - f(x)}{h} = 2x \)
3. \( f(x+h) - f(x) = 2xh \)
4. \( f(x+h) = f(x) + 2xh \)

Convert to code

```fortran
REAL*8 H,X,F(20)
INTEGER I

H = 0.01
F(1) = 1
DO I = 1, 19
   X = 1.0 + I*H
   F(I+1) = F(I) - 2*H
ENDDO
```
Writing Scientific Programs by Hand

Derive equations

Convert to code

Problems:

- Transcription errors
- Identifying error from testing final program
Any problem in computer science can be solved with another layer of indirection.

David Wheeler

I’d rather write programs to write programs than write programs

Richard Sites

Computational Thinking - The thought processes involved in formulating problems so their solutions can be represented as computational steps and algorithms.

Alfred Aho
Components of a Program to Write Scientific Programs

- Description of problem
  - Domain Specific Language
  - Symbolic mathematics
- Transformation to target
- Representation of target language/system
Other Projects

- FEniCS - Finite element solutions to differential equations
- SAGA (Scientific computing with Algebraic and Generative Abstractions) - PDE’s
- Spiral - signal processing transforms
- TCE (Tensor Contraction Engine) - quantum chemistry
- FLAME (Formal Linear Algebra Method Environment) - Linear algebra

See Andy Terrel’s article in CiSE March/April 2011
Advantages and Disadvantages

▶ Advantages
  ▶ Improved notation for expressing problems and algorithms
  ▶ Testability - transforms are ‘ordinary software’
  ▶ Optimization of generated code
    ▶ Domain specific optimizations
    ▶ Explore larger parameter space
    ▶ Restructuring for various target systems

▶ Disadvantages
  ▶ If problem domain isn’t covered by existing project, ?
Outline

Motivation and Overview of Writing Scientific Programs

Implementation of a Framework

Example: Partition Function Integral
Implementing Components of a Program to Write Scientific Programs

- Description of problem
  - Symbolic mathematics - SymPy expressions
  - Structure above expressions - derivation modeling
- Transformation to target - pattern matching
- Representation of target language/system - classes for C++ and Python
Derivation Modeling - What is it?

Think of math homework

- Series of steps
- Show your work

Solve for $x$:

$$2x + y = 44$$
$$2x = 44 - y$$
$$x = 22 - y/2$$

Types of steps

- Exact transformations
- Approximations
- Specialization - no. of spatial dimensions, no. of particles
Derivation Modeling

derivation class
  ► constructor takes initial equation
  ► add_step
  ► final or new_derivation

Examples of steps:
  ► replace
  ► add_term
  ► specialize_integral

Also outputs steps to web page in MathML or MathJax for nicely rendered math.
Derivation Modeling - Example

```python
from sympy import Symbol, S
from prototype.derivation import derivation, add_term, mul_factor

x, y = Symbol('x'), Symbol('y')
d = derivation(2*x+y, 44)
d.add_step(add_term(-y), 'Subtract y')
d.add_step(mul_factor(S.Half), 'Divide by 2')
print(d.final())
```

Output: \[ x = -\frac{y}{2} + 22 \]
from sympy import Symbol, print_tree
x,y = Symbol('x'), Symbol('y')
e = x+y
print_tree(e)

Add: x + y
+ Symbol: y
| comparable: False
+ Symbol: x
  comparable: False
Transform to Target System - Pattern Matching

Add: $x + y$

Symbol: $y$
comparable: False

Symbol: $x$
comparable: False

Match SymPy expression in Python

```python
v = AutoVar()
m = Match(e)
if m(Add, v.e1, v.e2):
    # operate on v.e1 and v.e2
```
object.__getattr__(self,name)
If attribute not found, this method is called

```python
class AutoVar(object):
    def __init__(self):
        self.vars = []
    def __getattr__(self,name):
        self.vars.append(name)
        return AutoVarInstance(self,name)
```
def expr_to_py(e):
    v = AutoVar()
    m = Match(e)
    # subtraction
    if m(Add, (Mul, S.NegativeOne, v.e1), v.e2):
        return py_expr(py_expr.PY_OP_MINUS, expr_to_py(v.e2),
                       expr_to_py(v.e1))
    # addition
    if m(Add, v.e1, v.e2):
        return py_expr(py_expr.PY_OP_PLUS, expr_to_py(v.e1),
                       expr_to_py(v.e2))
    # division
    if m(Mul, v.e1, (Pow, v.e2, S.NegativeOne)):
        return py_expr(py_expr.PY_OP_DIVIDE, expr_to_py(v.e1),
                       expr_to_py(v.e2))
Approaches to Code Generation

- Print target as string
  
  ```python
  print "print 'Hello' "
  ```

- General (text-based) templating

- Structured model of target language and system
  
  ```python
  py_print_stmt(py_string("Hello"))
  ```
Overview of workflow

Input file

```python
r1 = Vector('r1', dim=2)
r2 = Vector('r2', dim=2)

V2 = Function('V')
n2 = partition_function.new_derivation()
n2.set_name('specialize_n2d2')
n2.set_title('Specialized to N=2, D=2')

n2.add_step(specialize_integral(R,(r1,r2)),'

r_cm = Vector('r_cm',dim=2)
r_12 = Vector('r_12',dim=2)
```

HTML + MathJax

Specialized to N=2, D=2

Derivation

\[ Z = \int e^{-\sqrt{s}} dR \]

specialize to N=2

\[ Z = \int \int e^{-\sqrt{s}} d_1 d_2 \]

Python

```python
import math
from ptrap_gen import trap0,trap1
def f_r_12_x(r_12_x):
    v = trap1(-1.9999999999999, 2.000000)
    return v

def f_r_12_y(r_12_x,r_12_y):
    eps = 1e-20
    return math.exp(-4.0 * ((r_12_y)**2 +
    n = 1000
    v = 16.0 * trap0(-1.9999999999999, 2.000000)
    print v
```

C++

```c++
#include "ctrap_gen.h"
#include <math.h>
#include <stdio.h>

int n = 10;
double valOf_r_12_y(double r_12_x,double r_12_y):
    return exp(-4.0 * pow(pow(r_12_y,2)

double valOf_r_12_x(double r_12_x){
    double v = trap0(-1.9999999999999,2.000000)
    return v;
}

int main(){
```
Outline

Motivation and Overview of Writing Scientific Programs

Implementation of a Framework

Example: Partition Function Integral
Example from Statistical Mechanics

Partition Function Integral
- Derivation
- Code Generation
  - Python or C++

Quadrature Method - Trapezoidal Rule
- Derivation
- Code Generation
  - Python or C++

Existing Quadrature Library
Example from Statistical Mechanics

Partition function describes thermodynamics of a system

\[ Z = \text{Symbol}('Z') \]
\[ \text{partition\_function} = \text{derivation}(Z, \text{Integral}(\exp(-V/(k*T)), R)) \]

\[ Z = \int e^{-\frac{V}{Tk}} dR \]
Example from Statistical Mechanics 2

\[
Z = \int \int e^{-\beta V(r_1, r_2)} \, dr_1 dr_2
\]

n2.add_step(specialize_integral(R, (r1, r2)),
    "specialize to N=2")
n2.add_step(replace(V, V2(r1, r2)),
    "replace potential with N=2")
Example from Statistical Mechanics 3

\[ r_{\text{cm}} = \text{Vector}(\text{'}r_{\text{cm}}\text{',dim=2}) \]
\[ r_{12} = \text{Vector}(\text{'}r_{12}\text{',dim=2}) \]
\[ r_{12}\_\text{def} = \text{definition}(r_{12}, r_2-r_1) \]
\[ r_{\text{cm}}\_\text{def} = \text{definition}(r_{\text{cm}}, (r_1+r_2)/2) \]
\[ V_{12} = \text{Function}(\text{'}V\text{'}) \]
\[ n2.\text{add}\_\text{step}(\text{specialize}\_\text{integral}(r_1,(r_{12},r_{\text{cm}})), \text{'}\text{Switch variables}\text{'}) \]
\[ n2.\text{add}\_\text{step}(\text{replace}(V_{2}(r_1,r_2),V_{12}(r_{12})), \text{'}\text{Specialize to a potential that depends only on interparticle distance}\text{'}) \]
\[ n2.\text{add}\_\text{step}(\text{replace}(V_{12}(r_{12}),V_{12}(\text{Abs}(r_{12}))), \text{'}\text{Depend only on the magnitude of the distance}\text{'}) \]

\[ Z = \int \int e^{-\beta V(r_{12})} \, dr_{12}dr_{\text{cm}} \]
Integrate out $r_{cm}$, decompose into vector components and add integration limits

$$Z = L^2 \int_{-\frac{1}{2}L}^{\frac{1}{2}L} \int_{-\frac{1}{2}L}^{\frac{1}{2}L} e^{-\beta V(r_{x12}, r_{y12})} dr_{12x} dr_{12y}$$
Specialize to Lennard-Jones potential.

\[ V(r) = -\frac{4}{r^6} + \frac{4}{r^{12}} \]  

(1)

Insert values for box size, and temperature

\[ Z = 16.0 \int_{-2.0}^{2.0} \int_{-2.0}^{2.0} e^{\frac{1}{(r_{12x}^2 + r_{12y}^2)^{3/2}}} - 4.0 \frac{1}{(r_{12x}^2 + r_{12y}^2)^{6/2}} \, dr_{12x} dr_{12y} \]
<table>
<thead>
<tr>
<th>Method</th>
<th>Value</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>scipy.integrate.dblquad</td>
<td>285.97597</td>
<td>0.4</td>
</tr>
<tr>
<td>Trapezoidal rule (N=1000)</td>
<td>285.97594</td>
<td></td>
</tr>
<tr>
<td>Python</td>
<td></td>
<td>2.9</td>
</tr>
<tr>
<td>Shedskin (Python -&gt; C++)</td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>C++</td>
<td></td>
<td>0.5</td>
</tr>
</tbody>
</table>
Summary

More information at

http://quantum_mc.blogspot.com

Code available on GitHub

https://github.com/markdewing/sympy/tree/derivation_modeling/sympy/prototype
```python
from sympy import Symbol, Integral, exp, Function, Abs, Eq
from sympy.prototype.vector import Vector, VectorMagnitude
from sympy.prototype.vector_utils import decompose, add_limits, replace_func
from sympy.prototype.derivation import derivation, definition, replace_definition, specialize_integral, replace, do_integral, identity
from partition import partition_function, beta_def, R, V

r1 = Vector('r1', dim=2)
r2 = Vector('r2', dim=2)

V2 = Function('V')
n2 = partition_function.new_derivation()
n2.set_name('specialize_n2d2')
n2.set_title('Specialized to N=2, D=2')

n2.add_step(specialize_integral(R, (r1, r2)), 'specialize to N=2')
n2.add_step(replace(V, V2(r1, r2)), 'replace potential with N=2')

r_cm = Vector('r_cm', dim=2)
r_12 = Vector('r_12', dim=2)

r_12_def = definition(r_12, r_2-r1)
r_cm_def = definition(r_cm, (r1+r2)/2)

V12 = Function('V')```

Specialized to $N=2$, $D=2$

**Derivation**

\[
Z = \int e^{-\beta V} \, dR
\]

specialize to $N=2$

\[
Z = \int \int e^{-\beta V} \, dr_1 \, dr_2
\]

replace potential with $N=2$

\[
Z = \int \int e^{-\beta V_{(r_1, r_2)}} \, dr_1 \, dr_2
\]

Switch variables
import math
from ptrap_gen import trap0, trap1

def f_r_12_x(r_12_x):
    v = trap1(-1.99999999990000, 2.00000000000000, f_r_12_y, n_r_12_x)
    return v

def f_r_12_y(r_12_x, r_12_y):
    eps = 1e-20
    return math.exp(-4.0 * ((r_12_y**2 + r_12_x**2 + eps)**-6) + 4.0 * ((r_12_y**2 + r_12_x**2 + eps)**-3))

n = 10000
v = 16.0 * trap0(-1.99999999990000, 2.00000000000000, f_r_12_x, n)
print v

12,0-1

#include "ctrap_gen.h"
#include <math.h>
#include <stdio.h>

int n = 10;

double val0f_r_12_y(double r_12_x, double r_12_y) {
    return exp(-4.0 * pow(pow(r_12_y, 2) + pow(r_12_x, 2), -6) + 4.0 * pow(pow(r_12_y, 2) + pow(r_12_x, 2), -3));
}

double val0f_r_12_x(double r_12_x) {
    double v = trap1(-1.999999999990000, 2.000000000000000, val0f_r_12_y, n, r_12_x);
    return v;
}

int main() {
    double val0 = trap0(-1.999999999990000, 2.000000000000000, val0f_r_12_x, n);
    double v = 16.0 * val0;
    printf("val = \%g\n", v);
    return 0;
}