Julia for Linear Programming

Iain Dunning and Miles Lubin

MIT

IAP – January 16, 2013

What is Linear Programming (LP)

$$\begin{array}{rl} \text{Maximize} & c^T x \quad \text{over } x \in \mathbb{R}^N \\ \text{Subject to} & a_i^T x \leq b_i \quad \forall i = 1, \ldots, M \end{array}$$

- Widely used - airline scheduling, production planning, TeX hyphenation...
- Simplex algorithm for LP named in "Top 10 Algo. of 20th Century" by SIAM

How are they solved - Modeling

- Why use modeling languages?

```
maximize Obj:
  sum {j in 1..N} profit[j] * x[j];

subject to CapacityCons:
  sum {j in 1..N} weight[j] * x[j] <= Capacity;
```

- Options
  - Commercial - e.g. AMPL, GAMS
    - Specialized - so fast
    - Not general purpose language
  - Open-source - e.g. PuLP, Pyomo, CVX, YALMIP
    - Built on Python or MATLAB
    - Use operator overloading - slower

Modeling in Julia

`http://github.com/IainNZ/Julp`

- Julia replaces domain-specific language
- Use macros to avoid issues with operator overloading

```
m = Model("max")
x = [ Variable(m) for j = 1:N ]
profit = rand(N);  weight = rand(N);

setObjective(m,
    @sumExpr([ profit[j] * x[j] for j = 1:N ])
)

addConstraint(m,
    @sumExpr([ weight[j] * x[j] for j = 1:N ])
)
```

Macro and Benchmark

- Macro only 3 (long-ish) lines
- Breaks `[c[i] * x[i] for i = 1:N]` into...
    - `[c[i] for i = 1:N]`
    - `[x[i] for i = 1:N]`
- ... which is how constraints are stored
- Benchmark times (in seconds):

Lang.	N=5000	N=10000
AMPL	4	6
Julia (Julp)	6.44	16.29
PyPy2 (PuLP)	26.62	53.45
Python (PuLP)	111.80	222.95

How are they solved – Algorithms

- Dantzig's simplex algorithm most used method.
- Computationally very challenging to implement efficiently
    - Naturally not vectorizable – specialized sparse linear algebra
    - Typically memory bound – cache misses
- Matlab implementations too slow to be used in practice
    - High-quality open-source codes exist in C++
- Can Julia compete?

Simplex Benchmarks

`https://github.com/mlubin/SimplexBenchmarks`

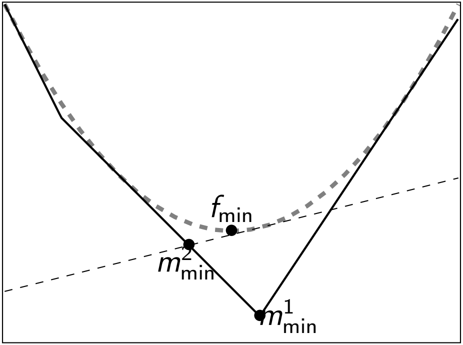
- Benchmark of some important operations:

	Julia	C++	C++bnd	Matlab	PyPy	Python
Sp. mat-sp. vec	1.29	0.90	1.00	5.79	19.20	417.16
Sp. vector scan	1.59	0.96	1.00	13.98	13.81	48.39
Sp. axpy	1.85	0.70	1.00	19.12	9.21	78.65

- C++bnd = C++ with bounds checking
- Execution times relative to C++bnd

Tutorial

`https://github.com/JuliaLang/IAP2013/blob/master/NumericalOptimization/tutorial.pdf`



- Tutorial for implementing a parallel asynchronous optimization algorithm in Julia using master-worker paradigm
- No background in optimization needed