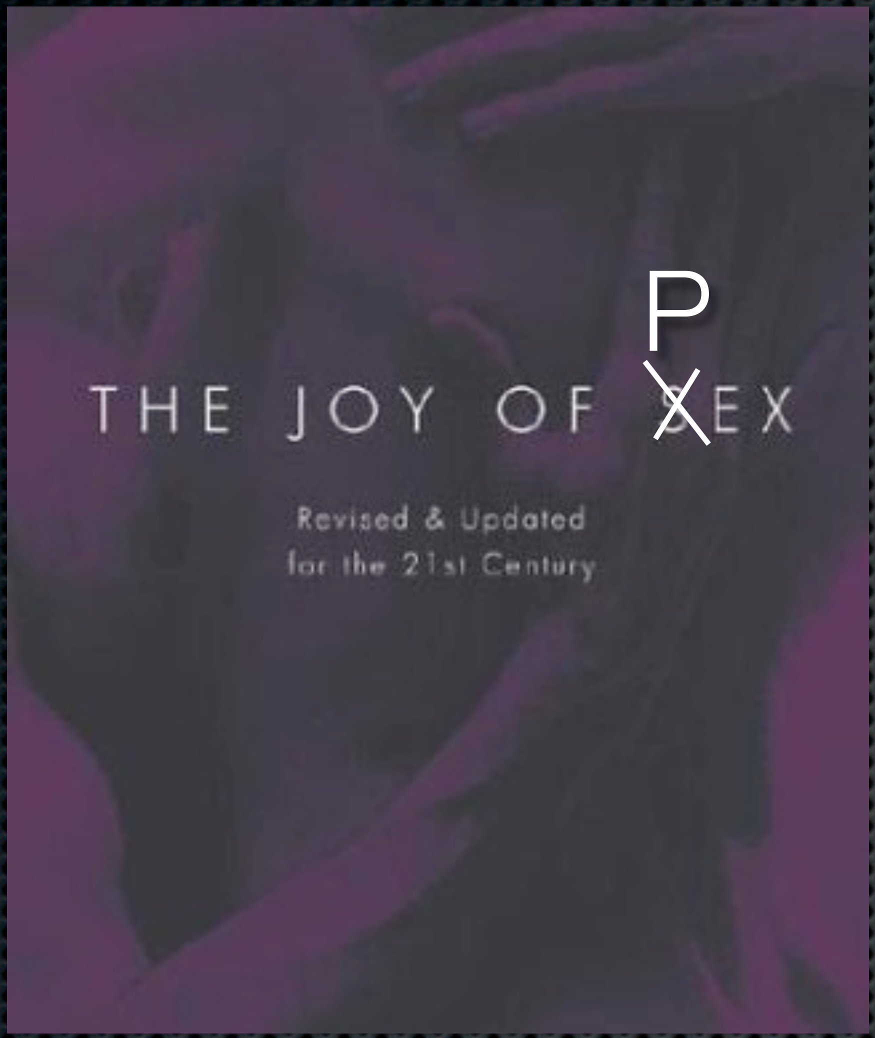


Lazy Man's Cython

Have Your Cake and Eat It Too

Dan Gindikin and Peter Yianilos

6/13/2008

PNYLAB LLC

What Is Pex?

Preprocessor and build system for Cython

What We Wanted

A language that gets down to the iron, runs at C speeds, and has no surprises in generated assembly, but at the same time guides you along to a clear, succinct and correct expression of complicated systems and algorithms.

What We Wanted More Concretely

PNYLAB LLC

What We Wanted More Concretely

1. Ineffable quality

What We Wanted More Concretely

1. Ineffable quality
    - write complicated algorithm

PNYLAB LLC

What We Wanted More Concretely

1. Ineffable quality
    - write complicated algorithm
    - either it is right the first time

What We Wanted More Concretely

1. Ineffable quality
    - write complicated algorithm
    - either it is right the first time
    - or it is very close, and easy to diagnose and fix

PNYLAB LLC

What We Wanted More Concretely

1. Ineffable quality
    - write complicated algorithm
    - either it is right the first time
    - or it is very close, and easy to diagnose and fix
2. Look at program assembly execution trace

PNYLAB LLC

What We Wanted More Concretely

1. Ineffable quality
    - write complicated algorithm
    - either it is right the first time
    - or it is very close, and easy to diagnose and fix
2. Look at program assembly execution trace
    - most instructions have to do with essence of problem

Real Problem for the World

PNYLAB LLC

Real Problem for the World

- Stay easy, friendly, interpreter-like

PNYLAB LLC

Real Problem for the World

- Stay easy, friendly, interpreter-like
- Yet use all the cycles the computer has to offer to solve the problem, not for overhead

PNYLAB LLC

Real Problem for the World

- Stay easy, friendly, interpreter-like
- Yet use all the cycles the computer has to offer to solve the problem, not for overhead
- Feel this has not been addressed, and not for any good technical reason

PNYLAB LLC

Discarded Candidates

Discarded Candidates

- No C, didn't feel right in the 21st century

PNYLAB LLC

Discarded Candidates

- No C, didn’t feel right in the 21st century
- No C++, didn’t think we were smart enough

Within a Stone's Throw

PNYLAB LLC

Within a Stone’s Throw

- Python, gets you everything except performance

PNYLAB LLC

Within a Stone’s Throw

- Python, gets you everything except performance
  - huge deal, wasn’t clear there could be a language that would corral you in the right direction

PNYLAB LLC

Within a Stone's Throw

- Python, gets you everything except performance
  - huge deal, wasn't clear there could be a language that would corral you in the right direction
- Pyrex, epsilon away, most of the heavy lifting done

Within a Stone's Throw

- Python, gets you everything except performance
  - huge deal, wasn't clear there could be a language that would corral you in the right direction
- Pyrex, epsilon away, most of the heavy lifting done
  - fast attribute access, exception handling, resource management - all the essentials for large system

Needful Things

PNYLAB LLC

Needful Things

- No gear shifting to C

Needful Things

- No gear shifting to C
- Stay Pythonic, see how far you can push it without sacrificing performance

Needful Things

- No gear shifting to C
- Stay Pythonic, see how far you can push it without sacrificing performance
- Naturally leads to a few desirables

Fast Numerics Essential?

PNYLAB LLC

Fast Numerics Essential?

- Already have linear algebra packages, but...

Fast Numerics Essential?

- Already have linear algebra packages, but...
- If Python API, Python overhead makes using small matrices infeasible

Fast Numerics Essential?

- Already have linear algebra packages, but...
- If Python API, Python overhead makes using small matrices infeasible
- May not have what you want

Fast Numerics Essential?

- Already have linear algebra packages, but...
- If Python API, Python overhead makes using small matrices infeasible
- May not have what you want
- Limits and contorts your thinking

PNYLAB LLC

Fast Numerics Essential?

- Already have linear algebra packages, but...
- If Python API, Python overhead makes using small matrices infeasible
- May not have what you want
- Limits and contorts your thinking
  - you jump through hoops to vectorize

PNYLAB LLC

Fast Numerics Essential?

- Already have linear algebra packages, but...
- If Python API, Python overhead makes using small matrices infeasible
- May not have what you want
- Limits and contorts your thinking
  - you jump through hoops to vectorize
  - a priori, you only consider things that are vectorizable

Fast Numerics Basic

PNYLAB LLC

Fast Numerics Basic

In Pyrex

```
cdef int i
arr=numpy.zeros(n)
for i from 0<=i<n:
    arr[i] = i
```

Fast Numerics Basic

In Pyrex

```
cdef int i
arr=numpy.zeros(n)
for i from 0<=i<n:
    arr[i] = i
```

Python speed

Fast Numerics
Basic

In Pyrex

```
cdef int i
arr=numpy.zeros(n)
for i from 0<=i<n:
    arr[i] = i
```

Python speed

In Pex

```
cdef int i
cdef ndarray<int, n> arr
```

PNYLAB LLC

Fast Numerics
Basic

In Pyrex

```
cdef int i
arr=numpy.zeros(n)
for i from 0<=i<n:
    arr[i] = i
```

Python speed

In Pex

```
cdef int i
cdef ndarray<int, n> arr
for i from 0<=i<n:
    arr{i} = i
```

Fast Numerics
Basic

In Pyrex

```
cdef int i
arr=numpy.zeros(n)
for i from 0<=i<n:
    arr[i] = i
```

Python speed

In Pex

```
cdef int i
cdef ndarray<int, n> arr
for i from 0<=i<n:
    arr{i} = i
```

C speed, as if arr is int*

PNYLAB LLC

Fast Numerics
Basic

In Pyrex

```
cdef int i
arr=numpy.zeros(n)
for i from 0<=i<n:
    arr[i] = i
```

Python speed

In Pex

```
cdef int i
cdef ndarray<int, n> arr
for i from 0<=i<n:
    arr{i} = i
```

C speed, as if arr is int*

Easily >100x faster

PNYLAB LLC

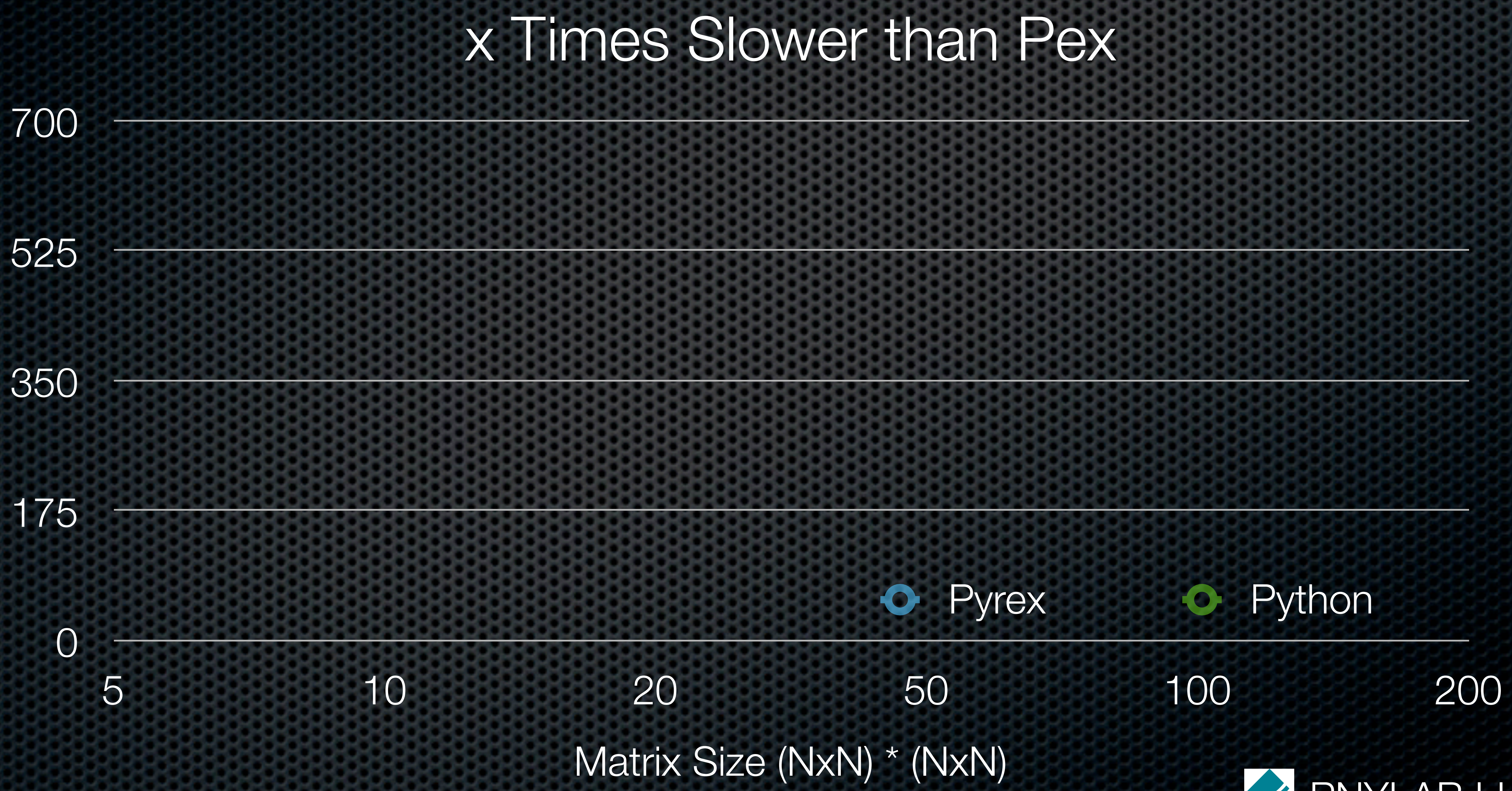
Matrix Multiply Pyrex

```
cdef void matmult(ndarray r,
        ndarray A, ndarray B):
    cdef int i,j,k
    for i from 0<=i<A.dimensions[0]:
        for j from 0<=j<B.dimensions[1]:
            for k from 0<=k<A.dimensions[1]:
                r[i,j]=r[i,j]+A[i,k]*B[k,j]
```

Matrix Multiply
Pex

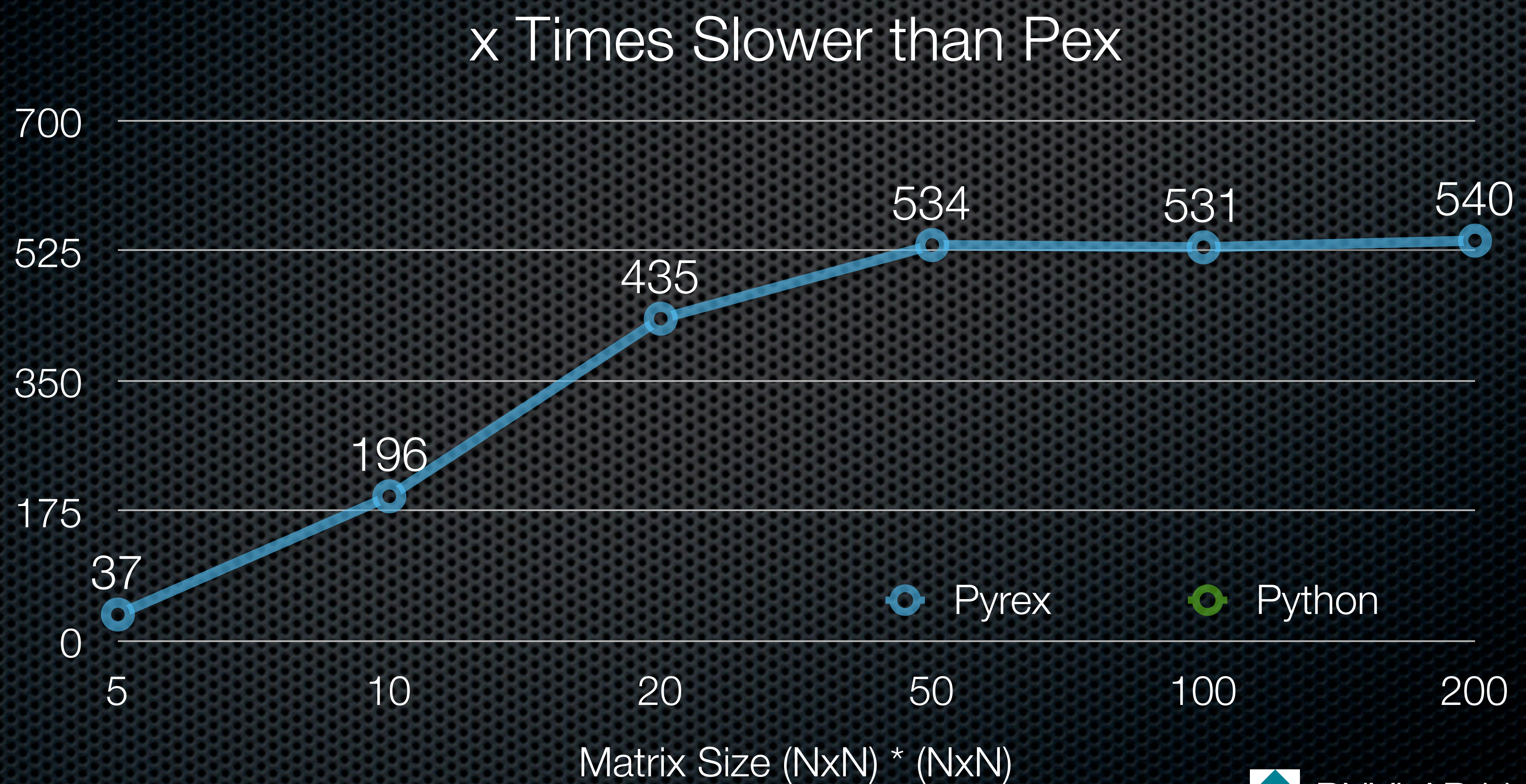
```
cdef void matmult(ndarray<double 2d> r,
        ndarray<double 2d> A, ndarray<double 2d> B):
    cdef int i,j,k
    for i from 0<=i<A.dimensions[0]:
        for j from 0<=j<B.dimensions[1]:
            for k from 0<=k<A.dimensions[1]:
                r{i,j}=r{i,j}+A{i,k}*B{k,j}
```

Matrix Multiply Performance



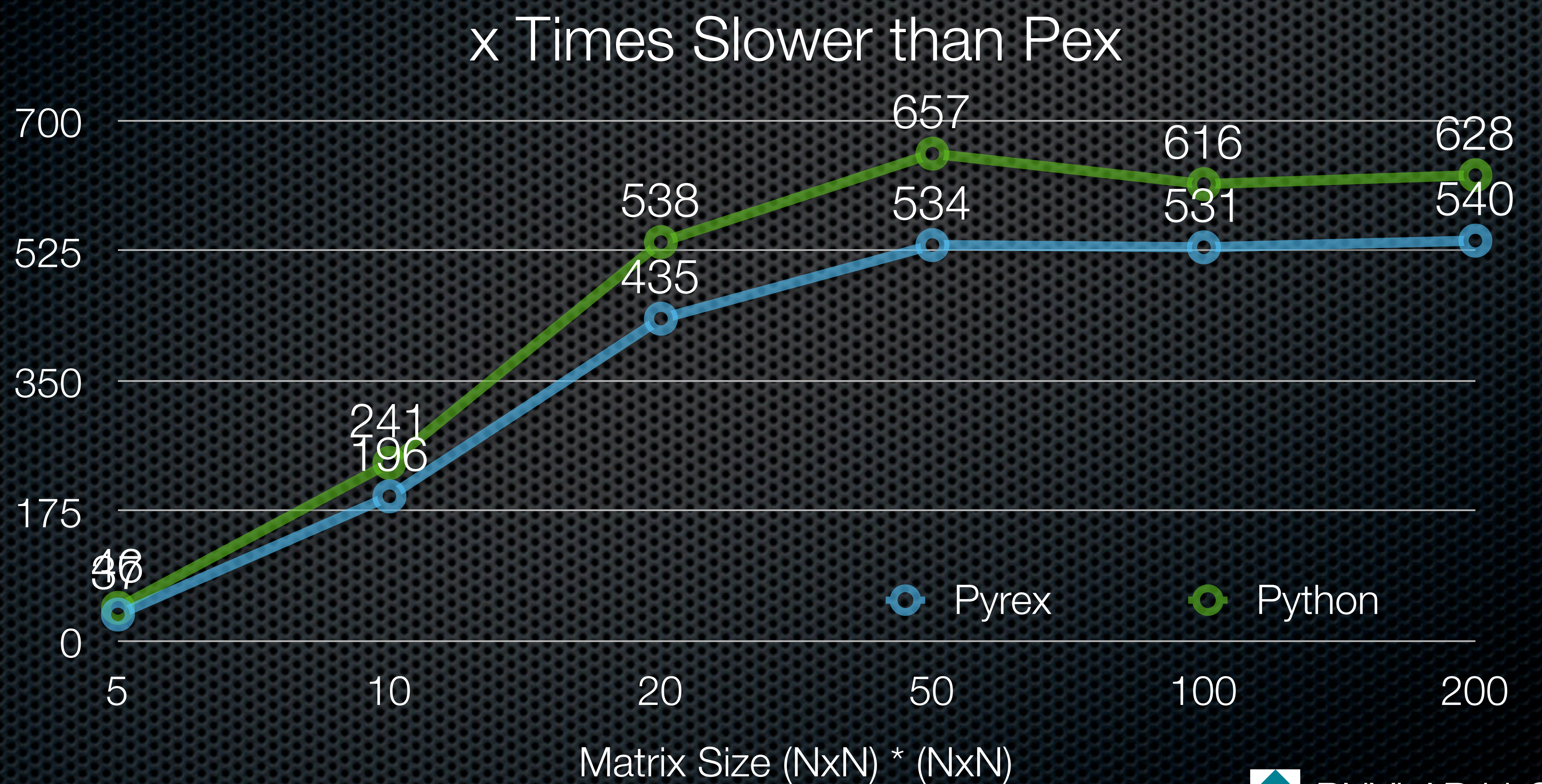
PNYLAB LLC

Matrix Multiply Performance



PNYLAB LLC

Matrix Multiply Performance



PNYLAB LLC

The Gauss-Jordan Sweep

$$\mathrm{H} = \mathrm{SWP[k]G}$$

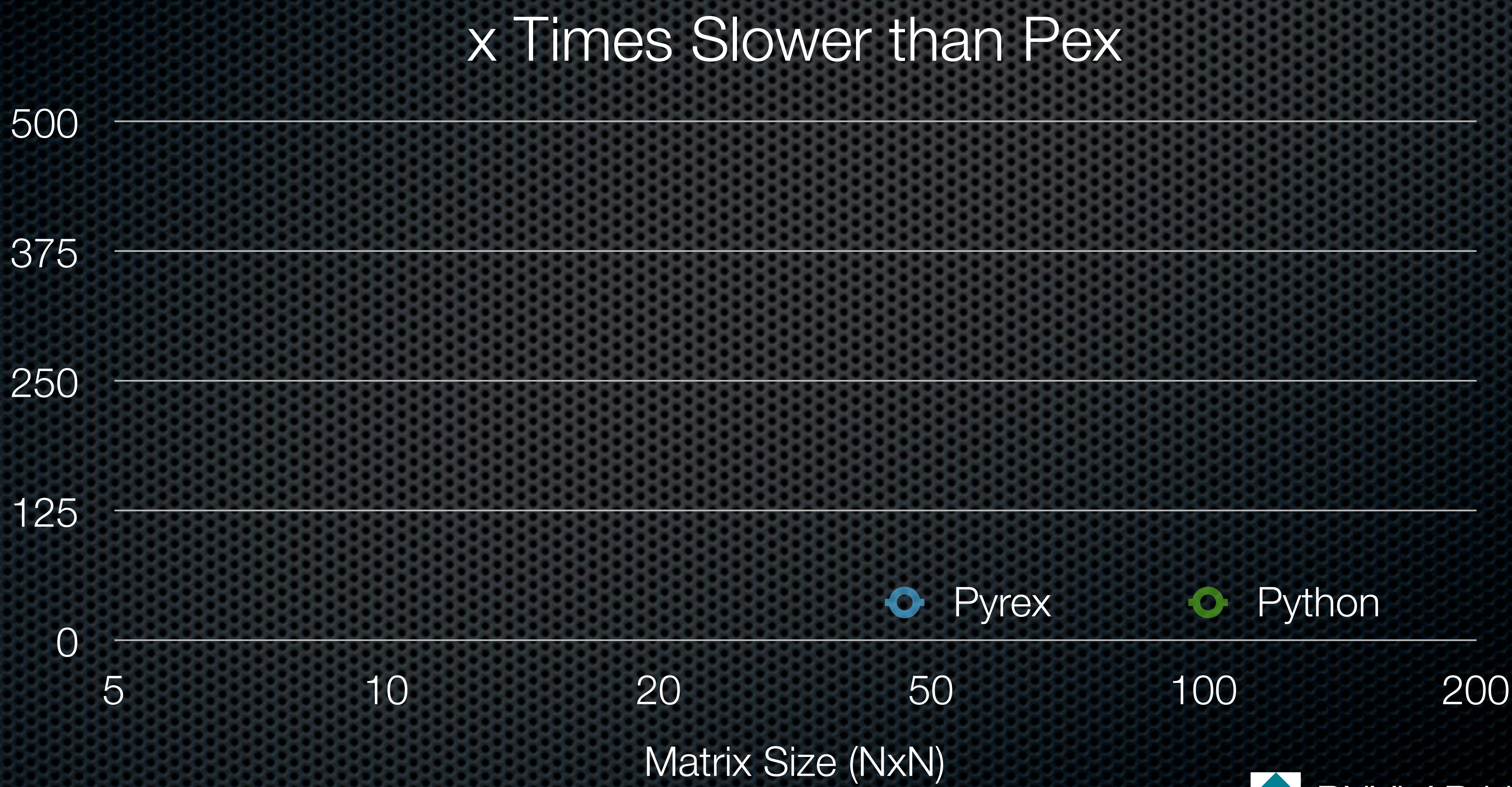
$$\mathrm{h}_{kk} = -1/g_{kk}$$
$$h_{jk} = h_{kj} = g_{jk}/g_{kk}, \quad j \neq k$$
$$h_{jl} = h_{lj} = g_{jl} - g_{jk}g_{kl}/g_{kk}, \quad j \neq k \text{ and } l \neq k$$

$$\mathrm{SWP}[1,2,\ldots,p]G = \begin{bmatrix} -G_{11}^{-1} & G_{11}^{-1}G_{12} \\ G_{21}G_{11}^{-1} & G_{22} - G_{21}G_{11}^{-1}G_{12} \end{bmatrix}.$$

```python
import numpy

def sweep(x):

    n = x.shape[0]



    l = 0
    u = n-1

    g_k=numpy.zeros(n,'double')

    pivot_product = 1.0

    for k in range(l,u+1):
        if x[k,k] == 0.0:
            pivot_product = 0.0
            break
        pivot_product *= x[k,k]
        x[k,k] = -1.0 / x[k,k]
        for j in range(n):
            if j == k:
                continue
            g_k[j] = x[j,k]
            x[j,k] = x[k,j] = -x[k,k] * g_k[j]
        for i in range(n):
            if i == k:
                continue
            for j in range(i+1):
                if j == k:
                    continue
                x[j,i] -= g_k[i] * x[k,j]
                x[i,j] = x[j,i]

    return pivot_product
```

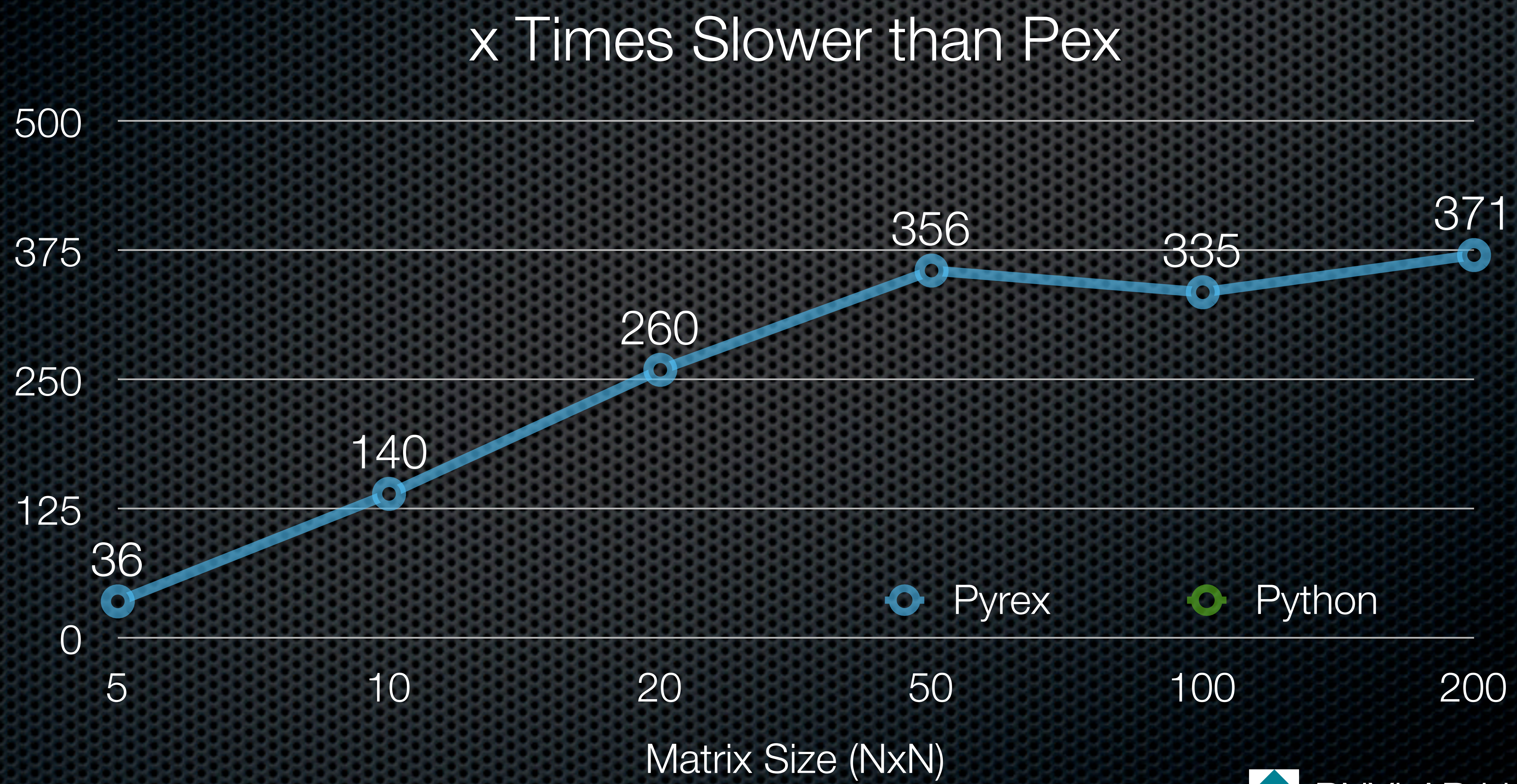
```cython
import numpy

def sweep(ndarray<double 2d> x):

    cdef int    n,l,u,i,j,k
    cdef double pivot_product

    n = x.dimensions[0]
    l = 0
    u = n-1

    cdef ndarray<double,n> g_k

    pivot_product = 1.0

    for k from l <= k <= u:
        if x{k,k} == 0.0:
            pivot_product = 0.0
            break
        pivot_product *= x{k,k}
        x{k,k} = -1.0 / x{k,k}
        for j from 0 <= j < n:
            if j == k:
                continue
            g_k{j} = x{j,k}
            x{j,k} = x{k,j} = -x{k,k} * g_k{j}
        for i from 0 <= i < n:
            if i == k:
                continue
            for j from 0 <= j <= i:
                if j == k:
                    continue
                x{j,i} -= g_k{i} * x{k,j}
                x{i,j} = x{j,i}

    return pivot_product
```

Sweep Algorithm Performance



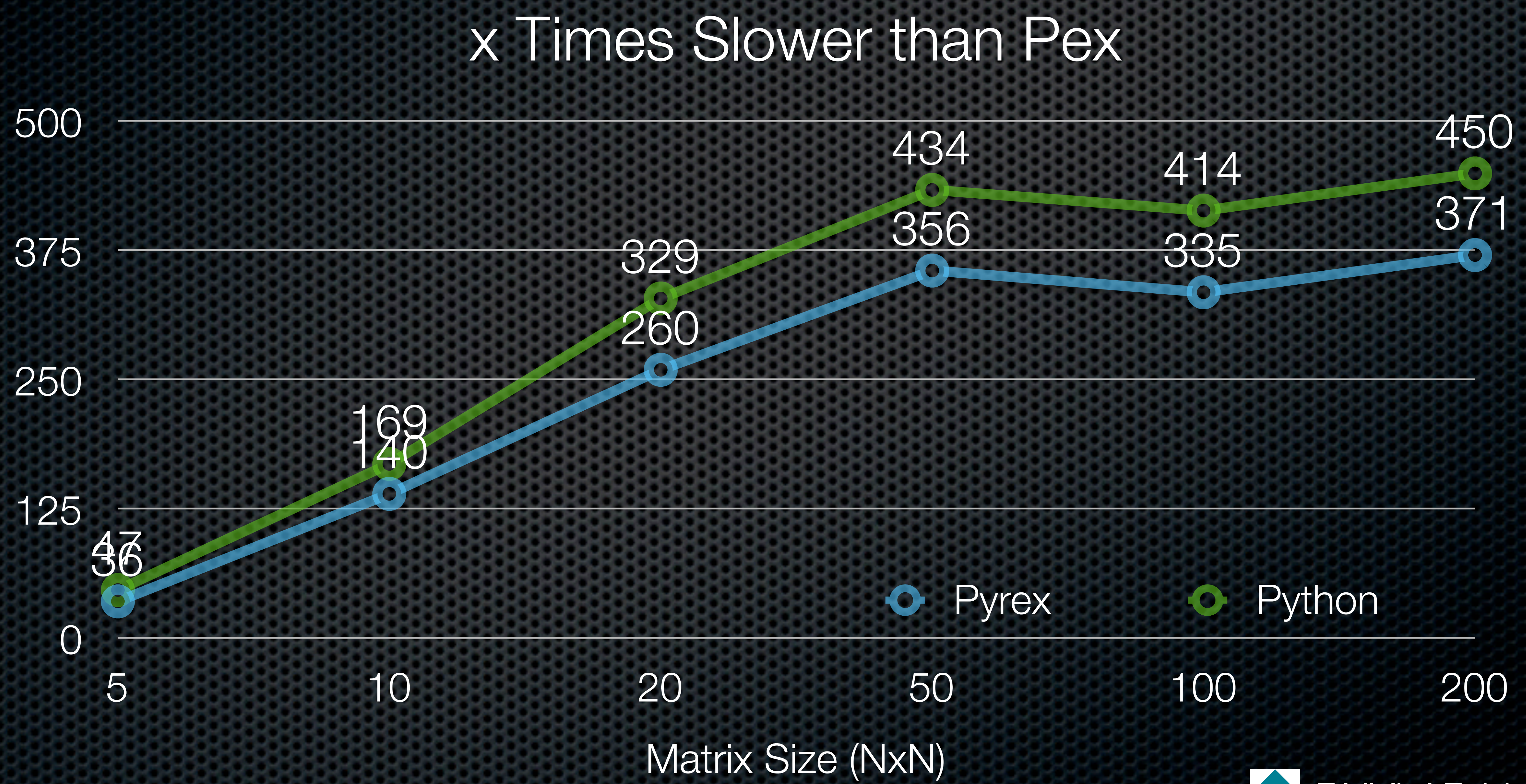
PNYLAB LLC

Sweep Algorithm Performance



PNYLAB LLC

Sweep Algorithm Performance



PNYLAB LLC

No Header Files

PNYLAB LLC

No Header Files

You write **file.px**

```
cdef class item:
    cdef double x,y,z
    cdef meth(me): pass
cdef func():pass
```

PNYLAB LLC

No Header Files

You write **file.px**

```
cdef class item:
    cdef double x,y,z
    cdef meth(me): pass
cdef func():pass
```

Pex produces **file.pxd**

```
cdef class item:
    cdef double x,y,z
    cdef meth(me)
cdef func()
```

PNYLAB LLC

No Header Files

You write **file.px**

```
cdef class item:
    cdef double x,y,z
    cdef meth(me): pass
cdef func():pass
```

Pex produces **file.pxd**

```
cdef class item:
    cdef double x,y,z
    cdef meth(me)
cdef func()
```

And **file.pyx**

```
<... implementation ...>
```

No Makefiles

PNYLAB LLC

No Makefiles

main.px

%pimport mod

No Makefiles

main.px

%pimport mod

mod.px

%pimport submod

PNYLAB LLC

No Makefiles

main.px

```
%pimport mod
```

mod.px

```
%pimport submod
```

submod.px

```
pass
```

PNYLAB LLC

No Makefiles

main.px

```
%pimport mod
```

mod.px

```
%pimport submod
```

submod.px

```
pass
```

- In the shell

```
$ pex main.px
```

PNYLAB LLC

No Makefiles

main.px

%pimport mod

mod.px

%pimport submod

submod.px

pass

- In the shell

  $ pex main.px

- Or in Python

  main=pex.pimport('main')

PNYLAB LLC

No Makefiles

main.px

```
%pimport mod
```

mod.px

```
%pimport submod
```

submod.px

```
pass
```

- In the shell

  $ pex main.px

- Or in Python

  main=pex.pimport('main')

- **submod** gets compiled, then **mod**, then **main**

No Makefiles

main.px

```
%pimport mod
```

mod.px

```
%pimport submod
```

submod.px

```
pass
```

- In the shell

  $ pex main.px

- Or in Python

  main=pex.pimport('main')

- **submod** gets compiled, then **mod**, then **main**
- Feels interpreted

Automatically pickleable cdef classes

Automatically pickleable cdef classes

- They are!
- Pex generates the magic __reduce__ and __setstate__ methods
- Caveat: can not have C pointer or struct attributes

PNYLAB LLC

Discovered We Wanted More

Fast Slices

PNYLAB LLC

Fast Slices

```
cdef ndarray<double,(n,m,k)> arr

arr{:,1:7,:-4}
```

PNYLAB LLC

Fast Slices

```
cdef ndarray<double,(n,m,k)> arr
```

```
arr{:,1:7,:-4}
```

same as

```
arr[:,1:7,:-4]
```

but does not plumb through python runtime, just quick creation of an ndarray header (in C code)

PNYLAB LLC

Faster Serialization

PNYLAB LLC

Faster Serialization

- Pickling

PNYLAB LLC

Faster Serialization

- Pickling
  - write
    - cdef class item: pass
    - pickle.dump(item(),open('file','w'))

PNYLAB LLC

Faster Serialization

- Pickling
  - write
    - cdef class item: pass
    - pickle.dump(item(),open('file','w'))
  - read
    - x = pickle.load(open('file'))

Faster Serialization

- Pickling
  - write
    - `cdef class item: pass`
    - `pickle.dump(item(),open('file','w'))`
  - read
    - `x = pickle.load(open('file'))`
- Goes through Python, slow

PNYLAB LLC

Faster Serialization - FastIO

PNYLAB LLC

Faster Serialization - FastIO

write

```
cdef item x = item()
x._fastdump_(open('file','w'))
```

Faster Serialization - FastIO

write

```
cdef item x = item()
x._fastdump_(open('file','w'))
```

read

```
x = pex_create_uninitialized(item)
x._fastload_(open('file'))
```

PNYLAB LLC

Faster Serialization - FastIO

write

```
cdef item x = item()
x._fastdump_(open('file','w'))
```

read

```
x = pex_create_uninitialized(item)
x._fastload_(open('file'))
```

>12x faster than pickling, as fast as writing a C struct

PNYLAB LLC

FastIO Limitations

PNYLAB LLC

FastIO Limitations

- Can’t dump a Python list

FastIO Limitations

- Can't dump a Python list
- Can't dump an ndarray of Python object

PNYLAB LLC

FastIO Limitations

- Can't dump a Python list
- Can't dump an ndarray of Python object
- All attributes must be either primitive C types (int, double, etc), or decorated ndarrays

FastIO Limitations

- Can’t dump a Python list
- Can’t dump an ndarray of Python object
- All attributes must be either primitive C types (int, double, etc), or decorated ndarrays
- This is just for the leafs of your object hierarchy

FastIO Limitations

- Can’t dump a Python list
- Can’t dump an ndarray of Python object
- All attributes must be either primitive C types (int, double, etc), or decorated ndarrays
- This is just for the leafs of your object hierarchy
- Still, can read/write mammoth data at C speed

Less Vigorous Coredump (1)

PNYLAB LLC

Less Vigorous Coredump (1)

Have **main.px**

```
cdef poof():
    cdef int *p=NULL
    p[0]
def func(): poof()
def main(): func()
```

PNYLAB LLC

Less Vigorous Coredump (1)

Have **main.px**

```
cdef poof():
    cdef int *p=NULL
    p[0]
def func(): poof()
def main(): func()
```

Guess what happens

```
$ pex main.px
```

PNYLAB LLC

Less Vigorous Coredump (2)

$ pex main.px

```
---- BEG BACKTRACE -----------
    Containing Executable File        Instruction Addr              Closest Symbol
    ./main.so                         0x3ACA                      __pyx_pf_201_func

    /usr/lib/libpython2.3.so.1.0      0x43991                      PyCFunction_Call
    /usr/lib/libpython2.3.so.1.0      0x20637                         PyObject_Call
    /usr/lib/libpython2.3.so.1.0      0x721B0         PyEval_CallObjectWithKeywords
    /usr/lib/libpython2.3.so.1.0      0x205FE                   PyObject_CallObject

    ./main.so                         0x37C3                      __pyx_pf_201_main

    /usr/lib/libpython2.3.so.1.0      0x780A6                     PyEval_EvalCodeEx
    /usr/lib/libpython2.3.so.1.0      0x7836D                       PyEval_EvalCode
    /usr/lib/libpython2.3.so.1.0      0x92952              PyRun_SimpleFileExFlags
    /usr/lib/libpython2.3.so.1.0      0x939A4                 PyRun_AnyFileExFlags
    /usr/lib/libpython2.3.so.1.0      0x9869E                               Py_Main

    python                            0x5B2                                    main

    /lib/tls/libc.so.6                0x14DE3                     __libc_start_main

    python                            0x501                                  (null)
                                                                            [START]
---- END BACKTRACE -----------
```

PNYLAB LLC

Bounds Checking

Bounds Checking

file **main.px**

```
cdef ndarray<int,n> arr
arr{n+1}
```

PNYLAB LLC

Bounds Checking

file **main.px**

```
cdef ndarray<int,n> arr
arr{n+1}
```

run with bounds checking (about 20 times slower)

Bounds Checking

file **main.px**

```
cdef ndarray<int,n> arr
arr{n+1}
```

run with bounds checking (about 20 times slower)

```
$ pex -b main.px
```

PNYLAB LLC

Bounds Checking

file **main.px**

```
cdef ndarray<int,n> arr
arr{n+1}
```

run with bounds checking (about 20 times slower)

```
$ pex -b main.px
```

```
Traceback (most recent call last):
  File "main.pyx", line 298, in main.main
    __px__ndarray_int_get1(arr,"arr",n+1,'n+1')   ##  arr{n+1} | main.px,4
IndexError: Out of bounds index access "n+1"==11 for dimension 1 of "arr" which has length 10
```

Compilation Configuration

PNYLAB LLC

Compilation Configuration

Setup link with external C libraries inside your **file.px**

```
%whencompiling:
    env.cc.append('-I../../vector/include')
    env.link.append('../../vector/vector.so')
```

Compilation Configuration

Setup link with external C libraries inside your **file.px**

```
%whencompiling:
    env.cc.append('-I../../vector/include')
    env.link.append('../../vector/vector.so')
```

Then bring in prototypes as usual

```
cdef extern from "vector.h": ...
```

General Pragma Mechanism

PNYLAB LLC

General Pragma Mechanism

```
def func(ndarray<int 2d> arr):
```

PNYLAB LLC

General Pragma Mechanism

```
def func(ndarray<int 2d> arr):
    %whencompiling:
        scope.pragma_ndarray_bounds_checks = True
```

General Pragma Mechanism

```
def func(ndarray<int 2d> arr):
    %whencompiling:
        scope.pragma_ndarray_bounds_checks = True
    arr{1,n+1}    # THROWS EXCEPTION
```

PNYLAB LLC

General Pragma Mechanism

```
def func(ndarray<int 2d> arr):
    %whencompiling:
        scope.pragma_ndarray_bounds_checks = True
    arr{1,n+1}    # THROWS EXCEPTION
```

Turns on bounds checks

General Pragma Mechanism

```
def func(ndarray<int 2d> arr):
    %whencompiling:
        scope.pragma_ndarray_bounds_checks = True
    arr{1,n+1}    # THROWS EXCEPTION
```

Turns on bounds checks

Works by scope, so here pragma applies only to func()

Conversion to and from Dictionaries

PNYLAB LLC

Conversion to and from Dictionaries

- cdef classes opaque to Python

Conversion to and from Dictionaries

- cdef classes opaque to Python
- Pex generates _todict_ and _fromdict_ methods

Conversion to and from Dictionaries

- cdef classes opaque to Python
- Pex generates _todict_ and _fromdict_ methods
- Define in Pex, **mod.px**

```
cdef class item:
    cdef int x,y
```

PNYLAB LLC

Conversion to and from Dictionaries

- cdef classes opaque to Python
- Pex generates _todict_ and _fromdict_ methods
- Define in Pex, **mod.px**

```
cdef class item:
    cdef int x,y
```

- From Python

```
mod=pex.pimport('mod')
x = mod.item()
```

PNYLAB LLC

Conversion to and from Dictionaries

- cdef classes opaque to Python
- Pex generates _todict_ and _fromdict_ methods
- Define in Pex, **mod.px**

```
cdef class item:
    cdef int x,y
```

- From Python

```
mod=pex.pimport('mod')
x = mod.item()
x._fromdict_({'x':7,'y':12})
```

Conversion to and from Dictionaries

- cdef classes opaque to Python
- Pex generates _todict_ and _fromdict_ methods
- Define in Pex, **mod.px**

```
cdef class item:
    cdef int x,y
```

- From Python

```
mod=pex.pimport('mod')
x = mod.item()
x._fromdict_({'x':7,'y':12})
print x._todict_()
```

out: {'y': 12, 'x': 7}

Gotchas

Gotchas

- Pex has no parser, regular expression based

PNYLAB LLC

Gotchas

- Pex has no parser, regular expression based
- Leads to annoying quirks, eg

```
def func(a, # comment
         b):
```

Gotchas

- Pex has no parser, regular expression based
- Leads to annoying quirks, eg

```
def func(a, # comment
         b):
```

- Joined to `def func(a, # comment b):`, so syntax error

PNYLAB LLC

Gotchas

- Pex has no parser, regular expression based
- Leads to annoying quirks, eg

```
def func(a, # comment
         b):
```

- Joined to `def func(a, # comment b):`, so syntax error
- Also, no real type system

PNYLAB LLC

Off the Reservation

PNYLAB LLC

Off the Reservation

- pointers (tool of the devil)
  - don’t use them
  - don’t think you need to
  - would like to prohibit them
- structs (use cdef classes instead)

Best of Both Worlds

PNYLAB LLC

Best of Both Worlds

- Luxuriate in Python decadence

Best of Both Worlds

- Luxuriate in Python decadence
    - lists, tuples, dicts, itertools, anything goes

Best of Both Worlds

- Luxuriate in Python decadence
  - lists, tuples, dicts, itertools, anything goes
  - most of the time

Best of Both Worlds

- Luxuriate in Python decadence
  - lists, tuples, dicts, itertools, anything goes
  - most of the time
- Get down to the iron where it matters

Best of Both Worlds

- Luxuriate in Python decadence
  - lists, tuples, dicts, itertools, anything goes
  - most of the time
- Get down to the iron where it matters
  - not much additional pain, lots of performance

PNYLAB LLC

Big Picture
Lessons Learned

PNYLAB LLC

Big Picture Lessons Learned

- Have enough performance

PNYLAB LLC

Big Picture
Lessons Learned

- Have enough performance
  - INCREFd memory management - fast, good

Big Picture
Lessons Learned

- Have enough performance
  - INCREFd memory management - fast, good
- Compiler working hard not only OK, but what you want

Big Picture
Lessons Learned

- Have enough performance
  - INCREFd memory management - fast, good
- Compiler working hard not only OK, but what you want
- With this setup, someone who only knows Python, can write C efficient code

PNYLAB LLC

Big Picture
Lessons Learned

PNYLAB LLC

Big Picture
Lessons Learned

- **Coredumps change feel of language**

PNYLAB LLC

Big Picture
Lessons Learned

- **Coredumps change feel of language**
  - completely

PNYLAB LLC

Big Picture
Lessons Learned

- **Coredumps change feel of language**
  - completely
  - sleep worse

PNYLAB LLC

Big Picture
Lessons Learned

- **Coredumps change feel of language**
  - completely
  - sleep worse
  - waste life chasing down horrific memory bugs

Big Picture
Lessons Learned

- **Coredumps change feel of language**
  - completely
  - sleep worse
  - waste life chasing down horrific memory bugs
  - die younger

Big Picture
Lessons Learned

- **Coredumps change feel of language**
  - completely
  - sleep worse
  - waste life chasing down horrific memory bugs
  - die younger
  - taken away from essence of problem

Big Picture
Going Forward

Big Picture
Going Forward

- Control coredumps

Big Picture
Going Forward

- Control coredumps
- Give up pointers, naked memory access (Hello Fortran!)

Big Picture
Going Forward

- Control coredumps
- Give up pointers, naked memory access (Hello Fortran!)
  - not as horrible as it sounds

Big Picture
Going Forward

- Control coredumps
- Give up pointers, naked memory access (Hello Fortran!)
    - not as horrible as it sounds
    - have fast arrays, add in fast multiple value return

Big Picture
Going Forward

- Control coredumps
- Give up pointers, naked memory access (Hello Fortran!)
  - not as horrible as it sounds
  - have fast arrays, add in fast multiple value return
  - the only thing you give up: blitting

Big Picture
Going Forward

- Control coredumps
- Give up pointers, naked memory access (Hello Fortran!)
  - not as horrible as it sounds
  - have fast arrays, add in fast multiple value return
  - the only thing you give up: blitting
  - allows safe mode guaranteed to catch corruption

Going Forward
Safe Mode

Going Forward
Safe Mode

- Runs within 3-4x times slower

Going Forward
Safe Mode

- Runs within 3-4x times slower
- Guaranteed to catch any memory corruption

Going Forward
Safe Mode

- Runs within 3-4x times slower
- Guaranteed to catch any memory corruption
- Set a mask at compile time

PNYLAB LLC

Going Forward
Safe Mode

- Runs within 3-4x times slower
- Guaranteed to catch any memory corruption
- Set a mask at compile time
    - bounds checking

Going Forward
Safe Mode

- Runs within 3-4x times slower
- Guaranteed to catch any memory corruption
- Set a mask at compile time
  - bounds checking
  - uninitialized variable access

PNYLAB LLC

Going Forward
Safe Mode

- Runs within 3-4x times slower
- Guaranteed to catch any memory corruption
- Set a mask at compile time
  - bounds checking
  - uninitialized variable access
  - keeps track of object creation, detects leaked cycles

Wishlist - Performance

Wishlist - Performance

- Pragma C_code_only

Wishlist - Performance

- Pragma C_code_only
- Fast operator overloading

PNYLAB LLC

Wishlist - Performance

- Pragma C_code_only
- Fast operator overloading
- Fast multiple return

PNYLAB LLC

Wishlist - Performance

- Pragma C_code_only
- Fast operator overloading
- Fast multiple return
- Fast comprehensions: arr={i*i for i from 0<=i<n if i%2}

Wishlist - Performance

- Pragma C_code_only
- Fast operator overloading
- Fast multiple return
- Fast comprehensions: arr={i*i for i from 0<=i<n if i%2}
- Tool color codes source based on whether it’s C or Py

PNYLAB LLC

Wishlist - Comfort

Now	Want

PNYLAB LLC

Wishlist - Comfort

Now	Want
cdef ndarray<int,(3,4)> arr	cdef int arr{3,4}

PNYLAB LLC

Wishlist - Comfort

Now	Want
cdef ndarray<int,(3,4)> arr	cdef int arr{3,4}
cdef item x=item(arg1,arg2)	cdef item x(arg1, arg2)

PNYLAB LLC

Wishlist - Comfort

Now	Want
cdef ndarray<int,(3,4)> arr	cdef int arr{3,4}
cdef item x=item(arg1,arg2)	cdef item x(arg1, arg2)

And also want, efficient append to 1d ndarray

PNYLAB LLC

Status
Our Shop

PNYLAB LLC

Status
Our Shop

- 30 KLOC of Pex code (1.4 MLOC generated C)

Status
Our Shop

- 30 KLOC of Pex code (1.4 MLOC generated C)
- 5 people actively using Pex, more soon

Status
Our Shop

- 30 KLOC of Pex code (1.4 MLOC generated C)
- 5 people actively using Pex, more soon
- Business unforgiving, speed and quality essential

PNYLAB LLC

Status
Availability

- Python Software Foundation License (PSF)
- Works on Unix, Mac (all but coredump backtraces), Windows - probably close, but who knows
- Get
  - pexlang.sourceforge.net

Status
Immediate Future

- Want to stop heavy development for a year or so
- Happy to help move any features into Cython proper
- Happy to accept any patches

PNYLAB LLC

Conclusion

- Initial goal

  A language that gets down to the iron, runs at C speeds, and has no surprises in generated assembly, but at the same time guides you along to a clear, succinct and correct expression of complicated systems and algorithms.

- We feel we are there, and are prepared to live with rough edges for awhile

PNYLAB LLC

Implementation Details
Fast Numerics

cdef ndarray<int 2d> A	int *data = A.data
	int st0,st1
	st0 = A.strides[0]/sizeof(int)
	st1 = A.strides[1]/sizeof(int)
arr{i,j}	data[st0 * i + st1 * j]

Implementation Details
Build

PNYLAB LLC

Implementation Details
Build

```
        Root.px
        /     \
    A.px     B.px
   /    \        \
AA.px  AB.px     |
         |      /
       ABA.px
```

PNYLAB LLC

Implementation Details
Build

```
           Root.px
            /    \
        A.px      B.px
       /    \        \
  AA.px    AB.px     |
             |      /
           ABA.px
```

In Python could say

Root.A.AB.ABA.func

Implementation Details
Build

```
        Root.px
        /     \
    A.px      B.px
   /    \        \
AA.px   AB.px     |
          |      /
        ABA.px
```

In Python could say

Root.A.AB.ABA.func

In Pyrex, same thing!

PNYLAB LLC

Implementation Details
Build

```
        Root.px
        /     \
    A.px     B.px
   /    \        \
AA.px   AB.px    |
          |     /
        ABA.px
```

In Python could say

Root.A.AB.ABA.func

In Pyrex, same thing!

func could be cdef

Implementation Details
Build

```
          Root.px
          /     \
     A.px      B.px
    /    \        \
AA.px   AB.px     |
          |      /
        ABA.px
```

In Python could say

Root.A.AB.ABA.func

In Pyrex, same thing!

func could be cdef

Root must know all prototypes of **ABA** at compile time

Implementation Details
Build

PNYLAB LLC

Implementation Details

Build

```
        Root.px
        /     \
     A.px     B.px
    /    \       \
AA.px    AB.px    |
           |     /
         ABA.px*
```

PNYLAB LLC

Implementation Details
Build

```
        Root.px
        /     \
    A.px     B.px
   /    \        \
AA.px   AB.px     |
          |      /
        ABA.px*
```

If **ABA** changes

Implementation Details
Build

```
           Root.px
           /     \
       A.px     B.px
      /    \        \
  AA.px   AB.px     |
            |      /
          ABA.px*
```

If **ABA** changes

need **Root** recompile

PNYLAB LLC

Implementation Details
Build

```
        Root.px
        /     \
     A.px     B.px
     /   \       \
AA.px   AB.px    |
          |     /
        ABA.px*
```

If **ABA** changes

need **Root** recompile

must detect this before **Root** is imported, else it is too late

PNYLAB LLC

Implementation Details
Build

```
         Root.px
         /     \
      A.px      B.px
     /    \        \
 AA.px    AB.px     |
            |      /
          ABA.px*
```

If **ABA** changes

- need **Root** recompile
- must detect this before **Root** is imported, else it is too late

Must walk import tree in preorder