

## SIAM - Cython



- Cython is a (Pseudo-)Python to C compiler.

```
%cython
def sign(x):
    if x == 0:
        return 0
    elif x < 0:
        return -1
    else:
        return 1
```

[Users ro... 2 code sage4 spyx.c](#)

[Users ro...code sage4 spyx.html](#)

Cython provides

- automatic Python object memory management
- language extensions for statically declaring types
- easy conversion between C and Python types

```
%python

def f(x):
    return x*x

def intf_py(a, b, N):
    dx = (b-a)/N
    s = (f(a) + f(b))/2
    for k in range(1, N):
        s += f(a + k*dx)
    return s * dx

print intf_py(0, 1.0, 1000)
```

0.3333335

```
%cython
cdef double f(double x):
    return x*x

def intf_c(double a, double b, int N):
    """here is a docstring"""
    cdef double dx = (b-a)/N
    cdef double s = (f(a) + f(b))/2
```

```

cdef int k
for k in range(1, N):
    s += f(a + k*dx)
return s * dx

print intf_c(0, 1.0, 1000)

```

0.3333335

[Users\\_ro...2\\_code\\_sage36\\_spyx.c](#)    [Users\\_ro...ode\\_sage36\\_spyx.html](#)

intf\_c?

```

timeit("intf_c(0r, 1.0r, 1000r)")
625 loops, best of 3: 8.51  $\mu$ s per loop

```

```

timeit("intf_py(0r, 1.0r, 1000r)")
625 loops, best of 3: 571  $\mu$ s per loop

```

Cython can handle more complicated C types.

- structs, unions, enums
- arrays
- pointers (including function pointers)
- ...

```

%cython

cdef struct point_c:
    double x
    double y

cdef void average_c(point_c* res, point_c* pts, int n):
    res.x = res.y = 0
    cdef int i
    for i from 0 <= i < n:
        res.x += pts[i].x
        res.y += pts[i].y
    res.x /= n
    res.y /= n

cdef void (*func_pointer)(point_c*, point_c*, int)
func_pointer = average_c

cdef class Polygon:
    cdef int n
    cdef point_c* vs

    def __init__(self, vs):
        self.n = len(vs)
        self.vs = <point_c*>malloc(self.n * sizeof(point_c))
        for i from 0 <= i < self.n:
            self.vs[i].x, self.vs[i].y = vs[i]

    def __dealloc__(self):
        free(self.vs)

    def center(self):
        cdef point_c cen
        average_c(&cen, self.vs, self.n)

```

```

        return cen.x, cen.y

def plot(self):
    from sage.all import line
    n = self.n
    g = line([(self.vs[ 0 ].x, self.vs[ 0 ].y),
              (self.vs[n-1].x, self.vs[n-1].y)])
    for i from 0 <= i < n-1:
        g += line([(self.vs[ i ].x, self.vs[ i ].y),
                  (self.vs[i+1].x, self.vs[i+1].y)])
    return g

```

[Users\\_ro...2\\_code\\_sage11\\_spyx.c](#)

[Users\\_ro...ode\\_sage11\\_spyx.html](#)

Note:

- Full use of Sage (or any other Python library)
- Only need to optimize what you *want* to

```
type(Polygon)
```

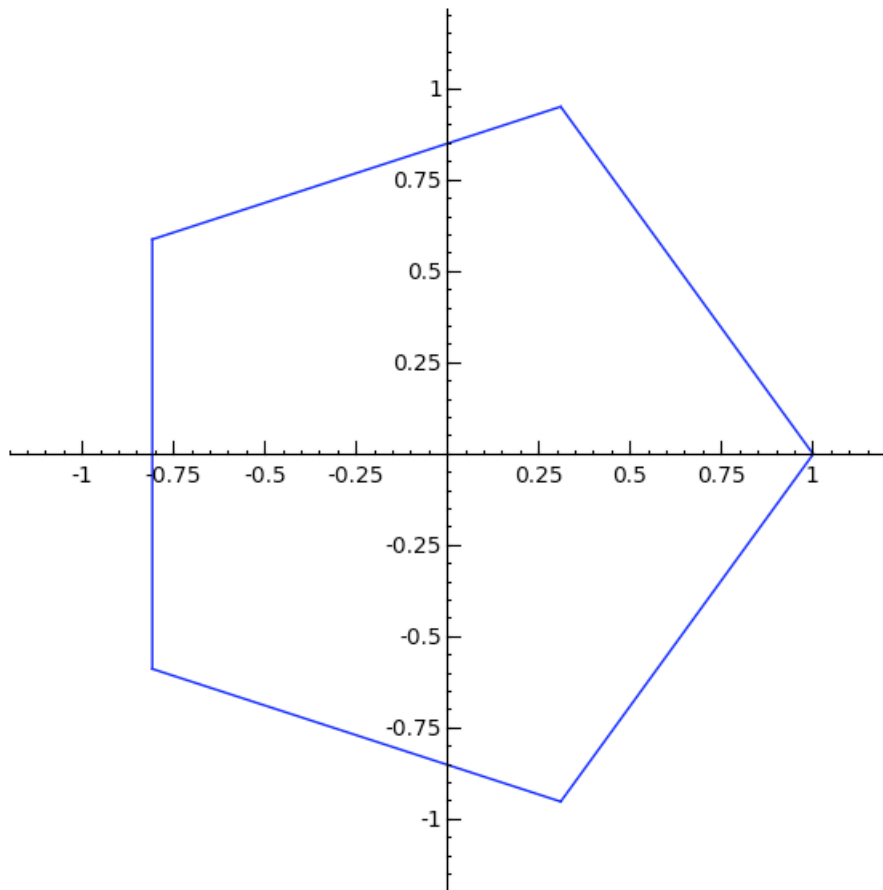
```
<type 'type'>
```

```

twopii = CDF(2*pi*I)
n = 5
vs = [exp(twopii*t/n) for t in range(n)]
p = Polygon([(v.real(), v.imag()) for v in vs])

```

```
p.plot().show(aspect_ratio=1)
```



```
p.center()
(-4.4408920985006264e-17, 2.2204460492503132e-17)
```

What about external code?

- Python is an excellent "glue" language
- Cython makes wrapping external code even easier

```
%cython
cdef extern from "gsl/gsl_poly.h":
    ctypedef struct gsl_poly_complex_workspace:
        size_t nc
        double* matrix

    gsl_poly_complex_workspace *gsl_poly_complex_workspace_alloc (size_t n)
    void gsl_poly_complex_workspace_free (gsl_poly_complex_workspace * w)

    int gsl_poly_complex_solve (double * a,
                                size_t n,
                                gsl_poly_complex_workspace * w,
                                double *z)

    int GSL_SUCCESS
```

```

def solve_poly(coeffs):
    coeffs = list(coeffs)
    cdef int i, n = len(coeffs)
    cdef double* a = <double*>malloc(n*sizeof(double))
    cdef double* z = <double*>malloc(2*n*sizeof(double))
    cdef gsl_poly_complex_workspace* workspace
    workspace = gsl_poly_complex_workspace_alloc(n)
    try:
        for i in range(n):
            a[i] = coeffs[i]
        gsl_poly_complex_solve(a, n, workspace, z)
        from sage.all import CDF
        I = CDF.gen()
        roots = [z[2*i] + z[2*i+1]*I for i from 0 <= i < n-1]
        return roots
    finally:
        free(a)
        free(z)
        gsl_poly_complex_workspace_free(workspace)

```

[Users ro...2 code sage18 spyx.c](#)

[Users ro...ode sage18 spyx.html](#)

```
R.<x> = QQ['x']
```

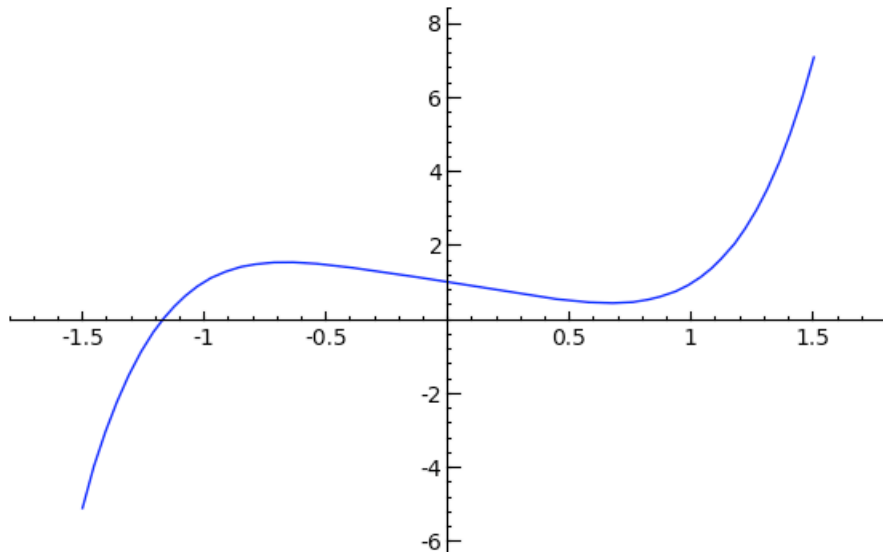
```
solve_poly(x^2-2)
```

```
[1.41421356237, -1.41421356237]
```

```
solve_poly(x^5-x+1)
```

```
[-1.16730397826, -0.18123244447 + 1.08395410132*I, -0.18123244447 -
1.08395410132*I, 0.764884433601 + 0.352471546032*I, 0.764884433601 -
0.352471546032*I]
```

```
plot(x^5-x+1, -1.5, 1.5)
```



```
f = (x-1)*(x-2)*(x-3)*(x-4)*(x^2+1)
```

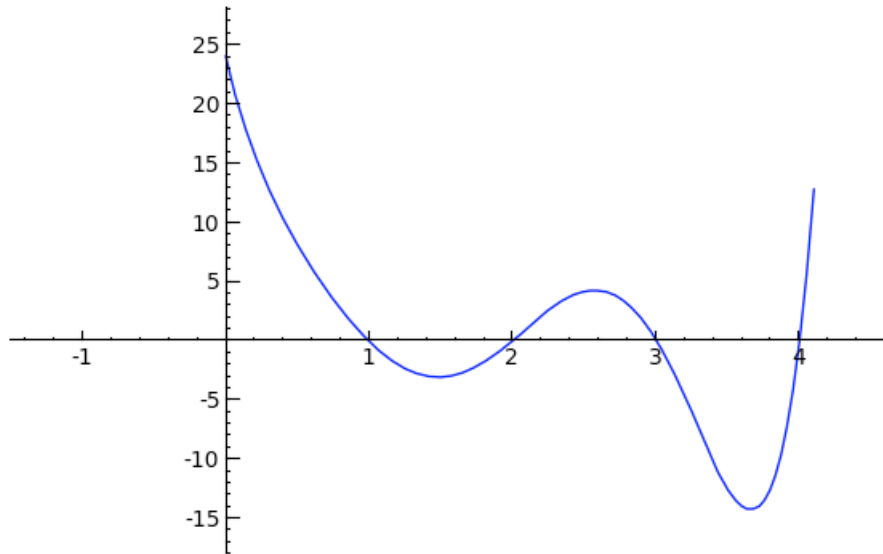
```
f
```

```
x^6 - 10*x^5 + 36*x^4 - 60*x^3 + 59*x^2 - 50*x + 24
```

```
solve_poly(f)
```

```
[3.33066907388e-16 + 1.0*I, 3.33066907388e-16 - 1.0*I, 1.0, 2.0,
3.0, 4.0]
```

```
plot(f, 0, 4.1)
```



```
f = ZZ['x'].random_element(degree=31); f
```

```
-10*x^31 - x^30 - x^29 + 16*x^28 + 2*x^26 + x^25 + x^24 - 3*x^22 -
10*x^21 + x^20 - 2*x^19 + 3*x^18 + 70*x^17 + x^16 - 17*x^15 + x^14 +
4*x^12 - x^11 - 7*x^10 - x^7 + 2*x^6 - 29*x^5 + 4*x^3 + 6*x^2 + 7*x
+ 13
```

```
vs = solve_poly(f); vs
```

```
[1.22976500956, 1.05823952338 + 0.400724682049*I, 1.05823952338 -
0.400724682049*I, 0.924848300404 + 0.12437085671*I, 0.924848300404 -
0.12437085671*I, 0.808778918273 + 0.516386468473*I, 0.808778918273 -
0.516386468473*I, 0.684621737176 + 0.871004586504*I, 0.684621737176
- 0.871004586504*I, 0.527325116538 + 0.795240210126*I,
0.527325116538 - 0.795240210126*I, 0.196507884935 + 1.09167965359*I,
0.196507884935 - 1.09167965359*I, 0.178459701673 + 0.771713617386*I,
0.178459701673 - 0.771713617386*I, -0.342106381829 +
1.14519752818*I, -0.342106381829 - 1.14519752818*I, -0.0518896023021
+ 0.911416360314*I, -0.0518896023021 - 0.911416360314*I,
-0.750030920391 + 1.04825434467*I, -0.750030920391 -
1.04825434467*I, -0.455445439776 + 0.824528087976*I, -0.455445439776
- 0.824528087976*I, -1.0117800921 + 0.0749800653699*I, -1.0117800921
- 0.0749800653699*I, -0.943746581348 + 0.542002837147*I,
-0.943746581348 - 0.542002837147*I, -0.839861640411 +
0.414273353096*I, -0.839861640411 - 0.414273353096*I,
-0.648803028999 + 0.463895615481*I, -0.648803028999 -
0.463895615481*I]
```

```
[abs(f(v)) for v in vs]
```

```
[5.65396618413e-11, 9.11555131877e-12, 9.11555131877e-12,
3.93470057076e-13, 3.93470057076e-13, 2.5063790708e-13,
2.5063790708e-13, 2.64917464257e-12, 2.64917464257e-12,
6.81493924654e-13, 6.81493924654e-13, 1.25707397213e-11,
1.25707397213e-11, 1.17444595141e-13, 1.17444595141e-13,
2.32978312407e-11, 2.32978312407e-11, 2.29468247395e-13,
2.29468247395e-13, 4.74151877861e-11, 4.74151877861e-11,
4.23153404465e-13, 4.23153404465e-13, 2.03669290327e-12,
2.03669290327e-12, 1.694525252e-12, 1.694525252e-12,
3.89585469653e-14, 3.89585469653e-14, 1.42108547152e-14,
1.42108547152e-14]
```

```
sum(point((v.real(), v.imag())) for v in vs)
```

