공역 구배법(conjugate gradient method)

학번	이름
175781	장석민

4개의 샘플, 8개의 방법

scipy.sparse.linalg의 함수들은 CSC방법으로 구성한 행렬을 사용하였다.

본문 예제 5번

3 x 3 | 7 entries | 22.2% sparsity

$$A = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{bmatrix}, \quad x = \begin{bmatrix} -1 \\ 2 \\ 3 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 8 \\ -5 \end{bmatrix}$$

방법	실행시간(1000회 평균)	결과 오차(절대값 오차 합)
scipy.linalg.solve	0.0615 ms	2.22e-16
cg in book	0.1560 ms	9.99e-16
cg in wikipedia	0.0853 ms	9.99e-16
scipy.sparse.linalg.cg	0.1890 ms	9.99e-16
scipy.sparse.linalg.cgs	0.2820 ms	1.11e-15
scipy.sparse.linalg.bicg	0.3580 ms	9.99e-16
scipy.sparse.linalg.bicgstab	0.2430 ms	4.44e-16
scipy.sparse.linalg.spsolve	0.0494 ms	2.22e-16

위키백과 예제

2 x 2 | 4 entries | 0.00% sparsity

$$A = \begin{bmatrix} 4 & 1 \\ 1 & 3 \end{bmatrix}, \quad x = \begin{bmatrix} \frac{1}{11} \\ \frac{7}{11} \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

방법	실행시간(1000회 평균)	결과 오차(절대값 오차 합)
scipy.linalg.solve	0.0496 ms	0.0
cg in book	0.1180 ms	0.0
cg in wikipedia	0.0573 ms	0.0

방법	실행시간(1000회 평균)	결과 오차(절대값 오차 합)
scipy.sparse.linalg.cg	0.1540 ms	0.0
scipy.sparse.linalg.cgs	0.2350 ms	0.0
scipy.sparse.linalg.bicg	0.3030 ms	0.0
scipy.sparse.linalg.bicgstab	0.1500 ms	1.25e-16
scipy.sparse.linalg.spsolve	0.0478 ms	0.0

컴퓨터 연습문제 11번 (n = 1000)

1000 x 1000 | 3995 entries | 99.6% sparsity

$$A = \begin{bmatrix} 3 & -1 & & & & & & & & 0.5 \\ -1 & 3 & -1 & & & & & & 0.5 & \\ & -1 & 3 & -1 & & & & & & \\ & & \ddots & \ddots & \ddots & 0.5 & & & & \\ & & & -1 & 3 & -1 & & & & \\ & & & 0.5 & \ddots & \ddots & \ddots & & & \\ & & \iddots & & & -1 & 3 & -1 & & \\ & 0.5 & & & & & -1 & 3 & -1 & \\ 0.5 & & & & & & & -1 & 3 & \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad b = \begin{bmatrix} 2.5 \\ 1.5 \\ 1.5 \\ \vdots \\ 1.5 \\ 1.0 \\ 1.5 \\ \vdots \\ 1.5 \\ 1.5 \\ 2.5 \end{bmatrix}$$

방법	실행시간(1000회 평균)	결과 오차(절대값 오차 합)
scipy.linalg.solve	53.000 ms	0.461
cg in book	6.010 ms	0.461
cg in wikipedia	6.910 ms	0.461
scipy.sparse.linalg.cg	0.646 ms	0.461
scipy.sparse.linalg.cgs	0.645 ms	0.459
scipy.sparse.linalg.bicg	1.190 ms	0.461
scipy.sparse.linalg.bicgstab	0.651 ms	0.458
scipy.sparse.linalg.spsolve	0.604 ms	0.461

BCSSTK15

3948 x 3948 | 60882 entries | 99.6% sparsity

https://math.nist.gov/MatrixMarket/data/Harwell-Boeing/bcsstruc2/bcsstk15.html

$A = \text{BCSSTK15}, \quad x = \text{generated by scipy.random.rand}, \quad b = \text{calculated by scipy.matmul(A, x)}$

방법	실행시간(1000회 평균)	결과 오차(절대값 오차 합)
scipy.linalg.solve	797 ms	1.62e-10
cg in book	468 ms	708
cg in wikipedia	453 ms	708
scipy.sparse.linalg.cg	5100 ms*	4.51e+04*
scipy.sparse.linalg.cgs	76.1 ms	75.4
scipy.sparse.linalg.bicg	10200 ms*	1.23e+05*
scipy.sparse.linalg.bicgstab	58.7 ms	85.2
scipy.sparse.linalg.spsolve	74.2 ms	2.31e-10

***는 성능을 확인하기에 너무 오래 걸려 1회 실행시간을 기입함**

scipy.solve와 공역 구배법 비교 분석

작은 크기의 행렬에서 scipy.solve는 항상 다른 방법보다 빠르고 정확한 결과를 볼 수 있다. 하지만 거대한 선형 시스템에서는 공역 구배법 계열이 빠른 것을 알 수 있다. [본문 예제 5번]과 [컴퓨터 연습문제 11번]을 비교해보면 [본문 예제 5번]에서는 scipy.solve가 공역 구배법보다 3배 빠르며 4배 정확한 결과를 보여준다. 하지만 [컴퓨터 연습문제 11번]을 보면 공역 구배법이 scipy.solve보다 9배 빠르며 동일한 정확도의 결과를 보여준다. 4개의 샘플을 모두 고려하면 scipy.solve가 공역 구배법 계열보다 더 정확한 것을 볼 수 있다. 하지만 어느 정도 오차가 허용된다면 공역 구배법의 빠르기면에서 scipy.solve보다 적절한 방법이 됨을 알 수 있다.

수치해석학 바이블의 구현

```
import scipy as sp

def BookCG(A, b, e=1e-8, k_max=100):
    k = 0
    d_old = 0
    x = sp.zeros((A.shape[0]))
    r = b - sp.matmul(A, x)
    d = sp.dot(r, r)
    while sp.sqrt(d) > e*sp.sqrt(sp.dot(b, b)) and k < k_max:
        k = k + 1
        if k == 1:
            p = r
        else:
            beta = d/d_old
            p = r + beta*p
        w = sp.matmul(A, p)
        a = d/sp.dot(p, w)
        x = x + a*p
        r = r - a*w
```

```
        d_old = d
        d = sp.dot(r, r)

    return x
```

위키백과의 구현

```
import scipy as sp

def WikiCG(A, b, *, e=1e-8, iter_max=100):
    x = sp.zeros(b.shape, dtype=float)
    r = b - sp.matmul(A, x)
    p = sp.copy(r)
    rr = sp.dot(r, r)
    for _ in range(iter_max):
        Ap = sp.matmul(A, p)
        alpha = rr/sp.matmul(p, Ap)
        x += alpha*p
        r -= alpha*Ap
        if linalg.norm(r) < e:
            break
        beta = rr
        rr = sp.dot(r, r)
        beta = rr/beta
        p = r + beta * p
    return x
```

테스트 코드

```
self._test_cg('생략', linalg.solve, A, b, expect)
self._test_cg('생략', BookCG, A, b, expect)
self._test_cg('생략', WikiCG, A, b, expect)
self._test_cg('생략', ScipyCG, csc_matrix(A), b, expect)
self._test_cg('생략', ScipyCGS, csc_matrix(A), b, expect)
self._test_cg('생략', ScipyBicG, csc_matrix(A), b, expect)
self._test_cg('생략', ScipyBicGStab, csc_matrix(A), b, expect)
self._test_cg('생략', ScipySpSolve, csc_matrix(A), b, expect)
```

테스트 구현

```
import logging
import scipy as sp
from timeit import timeit

def _test_cg(self, tname, cg, A, b, expect):
    logging.info(f'{tname} with {cg.__name__}')
    expect = sp.sort(expect)
    actual = sp.sort(cg(A, b))
    logging.debug(f'{tname} A\n{str(A)}')
```

```
    logging.debug(f'{tname} b\n{str(b)}')
    logging.debug(f'{tname} expect\n{str(expect)}')
    logging.debug(f'{tname} actual\n{str(actual)}')
    elapsed = timeit(lambda: cg(A, b), number=1000)
    logging.info(f'{tname} {elapsed:.3}ms')
    abserrorsum = sp.sum(sp.absolute(expect - actual))
    logging.info(f'{tname} sum of absolute errors = {abserrorsum:.3}')
```