06a_linear_systems_iterative

October 16, 2019

1 Iterative methods for solving linear systems

Recall the prototypal PDE problem introduced in the Lecture 08:

$$-u_{xx}(x) = f(x) \quad \text{in } \Omega = (0,1)$$

$$u(x) = 0, \quad \text{on } \partial\Omega = \{0,1\}$$

For the numerical discretization of the problem, we consider a **Finite Difference (FD) Approximation**. Let n be an integer, a consider a uniform subdivision of the interval $(0,1)$ using n equispaced points, denoted by $\{x_i\}_{i=0}^{n}$. Moreover, let u_i be the FD approximation of $u(x_i)$, and similarly $f_i \approx f(x_i)$.

The linear system that we need to solve is

$$\begin{aligned} u_i &= 0 && i = 0, \\ \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2} &= f_i && i = 1, \dots, n-1, \\ u_i &= 0 && i = n. \end{aligned} \qquad (P)$$

[5]:
```
%matplotlib inline
from numpy import *
from matplotlib.pyplot import *

n = 33
h = 1./(n-1)

x=linspace(0,1,n)

a = -ones((n-1,)) # Offdiagonal entries
b = 2*ones((n,)) # Diagonal entries
A = (diag(a, -1) + diag(b, 0) + diag(a, +1))
A /= h**2
f = x*(1.-x)

# Change first row of the matrix A
A[0,:] = 0
A[:,0] = 0
```

```
A[0,0] = 1
f[0] = 0

# Change last row of the matrix A
A[-1,:] = 0
A[:,-1] = 0
A[-1,-1] = 1
f[-1] = 0

# Solution by direct method
u = linalg.solve(A, f)
```

1.1 Jacobi

$$x_i^{k+1} = \frac{1}{A_{ii}} \times \left(b_i - \sum_{j \neq i} a_{ij} x_j^k \right)$$

[6]:
```
def jacobi(A, b, nmax=10000, eps=1e-10):
    pass # TODO

sol_jacobi = jacobi(A, f)
print(linalg.norm(sol_jacobi - u)/linalg.norm(u))
```

```
4777 9.99278393347e-11
9.68209796752e-11
```

1.2 Gauss-Seidel

$$x_i^{k+1} = \frac{1}{A_{ii}} \times \left(b_i - \sum_{j=0}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} a_{ij} x_j^k \right)$$

[7]:
```
def gauss_seidel(A,b,nmax=10000, eps=1e-10):
    pass # TODO

sol_gauss_seidel = gauss_seidel(A, f)
print(linalg.norm(sol_gauss_seidel - u)/linalg.norm(u))
```

```
2390 9.9530299764e-11
9.57731380802e-11
```

Gradient method

$$\mathbf{r}^k = \mathbf{b} - A\mathbf{x}^k$$

$$\alpha^k = \frac{{\mathbf{r}^k}^T \mathbf{r}^k}{{\mathbf{r}^k}^T A \mathbf{r}^k}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{r}^k$$

Preconditioned gradient method

$$P\mathbf{z}^k = \mathbf{r}^k$$

$$\alpha^k = \frac{{\mathbf{z}^k}^T \mathbf{r}^k}{{\mathbf{z}^k}^T A \mathbf{z}^k}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{z}^k$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha^k A \mathbf{z}^k$$

[8]:
```
def gradient(A, b, P, nmax=8000, eps=1e-10):
    pass # TODO

sol_gradient = gradient(A, f, identity(len(A)))
print(linalg.norm(sol_gradient - u)/linalg.norm(u))
sol_preconditioned_gradient = gradient(A, f, A)
print(linalg.norm(sol_preconditioned_gradient - u)/linalg.norm(u))
```

```
3909 9.94753934189e-11
7.09530597579e-11
1 1.20640538134e-14
2.68805760723e-15
```

1.3 Conjugate gradient

$$\alpha^k = \frac{{\mathbf{p}^k}^T \mathbf{r}^k}{{\mathbf{p}^k}^T A \mathbf{p}^k}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k \mathbf{p}^k$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha^k A \mathbf{p}^k$$

$$\beta^k = \frac{(A\mathbf{p}^k)^T \mathbf{r}^{k+1}}{(A\mathbf{p}^k)^T \mathbf{p}^k}$$

$$\mathbf{p}^{k+1} = \mathbf{r}^{k+1} - \beta^k \mathbf{p}^k$$

Preconditioned conjugate gradient

$$\alpha^k = \frac{\mathbf{p}^{k^T}\mathbf{r}^k}{(A\mathbf{p}^k)^T\mathbf{p}^k}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha^k\mathbf{p}^k$$

$$\mathbf{r}^{k+1} = \mathbf{r}^k - \alpha^k A\mathbf{p}^k$$

$$P\mathbf{z}^{k+1} = \mathbf{r}^{k+1}$$

$$\beta^k = \frac{(A\mathbf{p}^k)^T\mathbf{z}^{k+1}}{\mathbf{p}^{k^T}A\mathbf{p}^k}$$

$$\mathbf{p}^{k+1} = \mathbf{z}^{k+1} - \beta^k\mathbf{p}^k$$

[13]:
```
def conjugate_gradient(A, b, P, nmax=len(A), eps=1e-10):
    pass # TODO

sol_conjugate_gradient = conjugate_gradient(A, f, identity(len(A)))
print(linalg.norm(sol_conjugate_gradient - u)/linalg.norm(u))
```

```
16 2.70422126826e-17
2.90303183784e-15
```