05a_linear_systems_direct

October 16, 2019

1 Direct methods for solving linear systems

Recall the prototypal PDE problem introduce in the Lecture 08:

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

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

The physical interpretation of this problem is related to the modelling of an elastic string, which occupies at rest the space $[0,1]$ and is fixed at the two extremes. The unknown $u(x)$ represents the displacement of the string at the point x, and the right-hand side models a prescribed force $f(x)$ on the string.

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 $f_i \approx f(x_i)$.

In order to formulate the discrete problem, we consider a FD approximation of the left-hand side, as follows:

$$-u_{xx}(x_i) \approx \frac{-u_{i-1} + 2u_i - u_{i+1}}{h^2}$$

being $h = \frac{1}{n-1}$ the size of each subinterval (x_i, x_{i+1}).

The problem that we need to solve is

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

Then, let us collect al the unknowns $\{u_i\}_{i=0}^n$ in a vector $\mathbf{u}$. Then, (P) is a linear system

$$A\mathbf{u} = \mathbf{f}.$$

In this exercise we will show how to use direct methods to solve linear systems, and in particular we will discuss the **LU** and **Cholesky** decompositions that you have studied in Lecture 07.

First of all, let use define n and $\{x_i\}_{i=0}^n$.

[32]:
```
%matplotlib inline
from numpy import *
from matplotlib.pyplot import *

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

x=linspace(0,1,n)
```

Let us define the left-hand side matrix A.

[33]:
```
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

print(A)
print(linalg.cond(A))
```

```
[[ 2048. -1024.     0. …,     0.     0.     0.]
 [-1024.  2048. -1024. …,     0.     0.     0.]
 [    0. -1024.  2048. …,     0.     0.     0.]
 …,
 [    0.     0.     0. …,  2048. -1024.     0.]
 [    0.     0.     0. …, -1024.  2048. -1024.]
 [    0.     0.     0. …,     0. -1024.  2048.]]
467.842628839
```

Moreover, let us choose

$$f(x) = x(1-x)$$

so that the solution $u(x)$ can be computed analytically as

$$u(x) = u_{\text{ex}}(x) = \frac{x^4}{12} - \frac{x^3}{6} + \frac{x}{12}$$

The right hand side **f** then is easily assembled as:

[34]:
```
f = x*(1.-x)
```

We still need to impose the boundary conditions at $x = 0$ and $x = 1$, which read

$$u_i = 0 \qquad\qquad i = 0,$$

and

$$u_i = 0 \qquad\qquad i = n,$$

These conditions are associated with the first (last, respectively) row of the linear system.

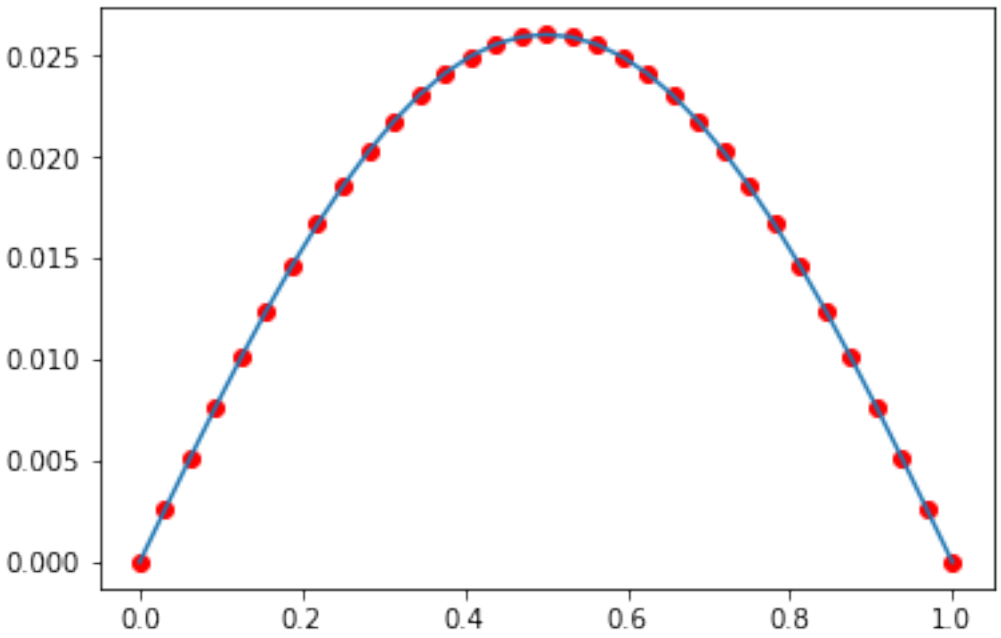
Then we can solve the linear system and compare the FD approximation of u to the exact solution u_{ex}.

[35]:
```
# 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

# Solve the linear system using numpy
A1 = A.copy()
u = linalg.solve(A1, f)
u_ex = (x**4)/12. - (x**3)/6. + x/12.

# Plot the FD and exact solution
_ = plot(x,u,'ro')
_ = plot(x,u_ex)
```



1.1 LU decomposition

We want to implement our linear solver using an **LU decomposition** (without pivoting)

$$A = LU$$

LU decomposition can be computed as in the following function.

[36]:
```
def LU(A):
    A = A.copy()
    N=len(A)
    for k in range(N-1):
        if (abs(A[k,k]) < 1e-15):
            raise RuntimeError("Null pivot")

        A[k+1:N,k] /= A[k,k]
        for j in range(k+1,N):
            A[k+1:N,j] -= A[k+1:N,k]*A[k,j]

    L=tril(A)
    for i in range(N):
        L[i,i]=1.0
    U = triu(A)
    return L, U

L, U = LU(A)
```

Once L and U have been computed, the system

$$A\mathbf{u} = \mathbf{f}$$

can be solved in **two steps**: first solve

$$L\mathbf{w} = \mathbf{f},$$

where L is a **lower triangular matrix**, and then solve

$$U\mathbf{u} = \mathbf{w}$$

where U is an **upper triangular matrix**.

These two systems can be easily solved by forward (backward, respectively) substitution.

[37]:
```
def L_solve(L,rhs):
    x = zeros_like(rhs)
    N = len(L)

    x[0] = rhs[0]/L[0,0]
    for i in range(1,N):
        x[i] = (rhs[i] - dot(L[i, 0:i], x[0:i]))/L[i,i]
```

```
    return x
```

[38]:
```
def U_solve(U,rhs):
    pass # TODO
```

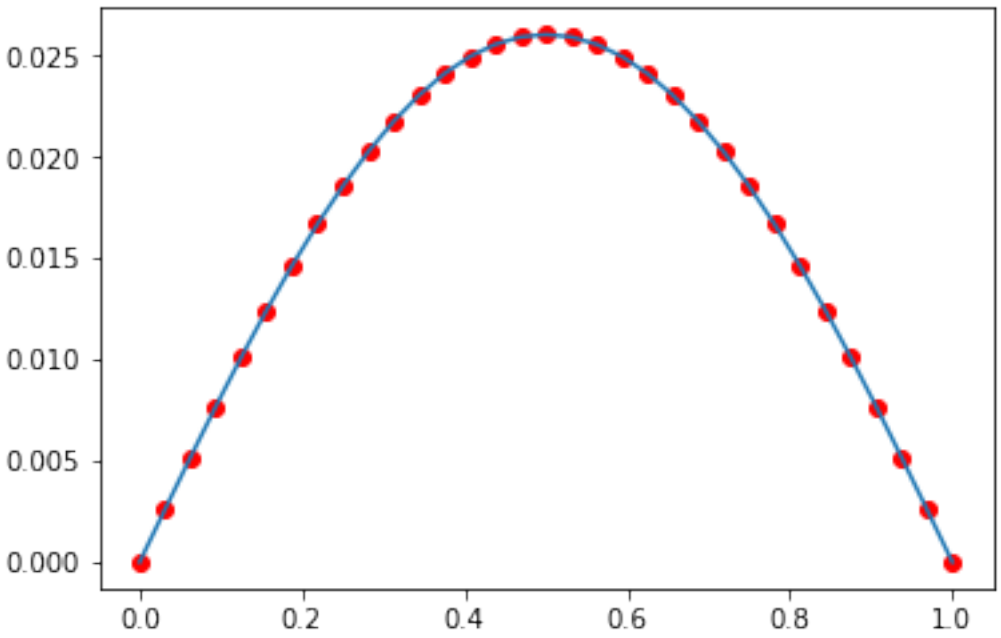
Now let's solve the system

$$A\mathbf{u} = \mathbf{f}$$

and compare the solution with respect to the exact solution.

[39]:
```
w = L_solve(L,f)
u = U_solve(U,w)

_ = plot(x,u,'ro')
_ = plot(x,u_ex)
```



1.1.1 try to compute the solution $u(x)$ with different forcing terms and compare with the exact solution without recomputing the LU decomposition

[40]:
```
# YOUR CODE HERE
```

1.2 Cholesky decomposition

For symmetric and positive define matrices, the Cholesky decomposition may be preferred since it reduces the number of flops for computing the LU decomposition by a factor of 2.

The Cholesky decomposotion seeks an upper triangular matrix H (with all positive elements on the diagonal) such that

$$A = H^T H$$

An implementation of the Cholesky decomposition is provided in the following function. We can use it to solve the linear system by forward and backward substitution.

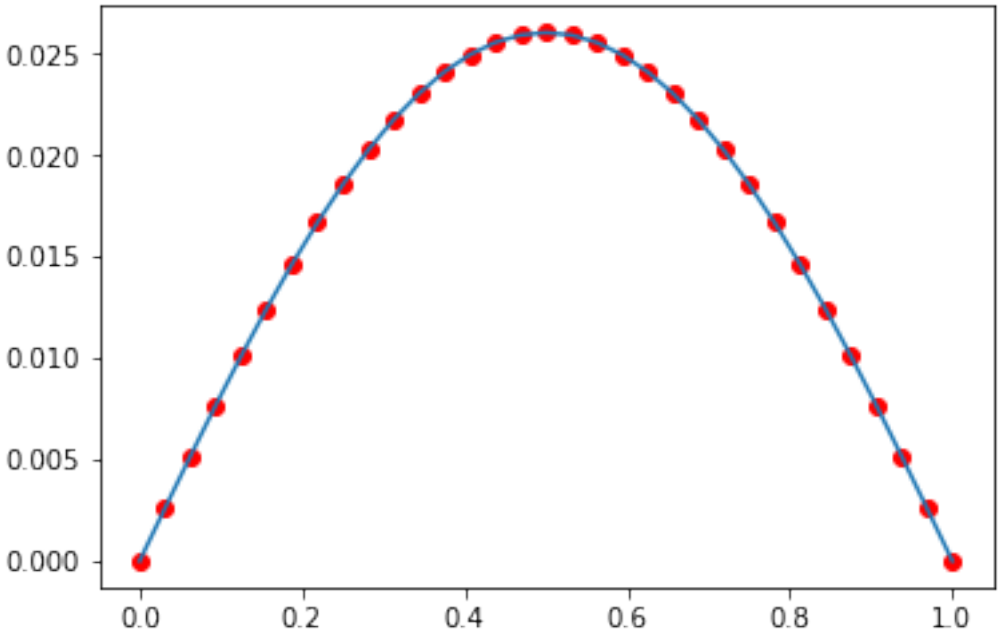
[41]:
```
def cholesky(A):
    A = A.copy()
    N = len(A)
    for k in range(N-1):
        A[k,k] = sqrt(A[k,k])
        A[k+1:N,k] = A[k+1:N,k]/A[k,k]

        for j in range(k+1,N):
            A[j:N,j] = A[j:N,j] - A[j:N,k]*A[j,k]

    A[-1,-1] = sqrt(A[-1,-1])
    L=tril(A)
    return L, L.transpose()


HT, H = cholesky(A)
y = L_solve(HT,f)
u = U_solve(H,y)


_ = plot(x,u,'ro')
_ = plot(x,u_ex)
```


0.025
0.020
0.015
0.010
0.005
0.000
0.0
0.2
0.4
0.6
0.8
1.0