CBE 502 - HW7 - Finite Elements

Tom Bertalan

December 25, 2012

Contents

List of Figures

1 Analytical Solution by Green's Function

1.1 Problem

$$\begin{aligned} L[u(x)] &= -f(x) \\ L[u(x)] &= \frac{\partial^2 u}{\partial x^2} \\ -f(x) &= A\sin(\omega x) + mx \end{aligned} \tag{1}$$

Boundary conditions:

$$\begin{aligned} u(x)\big|_{x=0} &= 1 \\ \left.\frac{\partial u(x)}{\partial x}\right|_{x=1} &= \epsilon \end{aligned} \tag{2}$$

Constants:

$$\begin{aligned} A &= 18.0 \\ \omega &= 10.0 \\ m &= 4.0 \\ \epsilon &= 0.5 \end{aligned} \tag{3}$$

Change of variables, to create homogeneous boundary conditions:

$$
\begin{aligned}
v &= u + ax + b \\
v' &= u' + a
\end{aligned} \tag{4}
$$

To make the new boundary conditions homogeneous, choose $b = -1$ and $a = -\epsilon$.

$$
-f(x) = L[u] = \frac{\partial^2 v}{\partial x^2} + 0 + 0 = L[v]
$$

That is, the problem does not change with the change-of-variables, only the boundary conditions.

1.2 Properties of the Green's Function

$G(x,t)$ satisfies the homogeneous problem (that is, $G''(x,t) = 0$):

$$
G(x,t) = \begin{Bmatrix} C_{1,1}x + C_{1,2}, & 0 \le x \le t \\ C_{2,1}x + C_{2,2}, & t \le x \le 1 \end{Bmatrix} = \begin{Bmatrix} C_{2,1}x + C_{2,2}, & 0 \le t \le x \\ C_{1,1}x + C_{1,2}, & x \le t \le 1 \end{Bmatrix} \tag{5}
$$

$G(x,t)$ satisfies homogeneous boundary conditions:

$$
\begin{aligned}
G(0,t) = 0 = C_{1,1} \cdot 0 + C_{1,2} &\rightarrow C_{1,2} = 0 \\
G'(0,t) = 0 = C_{2,1} \qquad &\rightarrow C_{2,1} = 0
\end{aligned} \tag{6}
$$

$G(x,t)$ is piecewise, but fully continuous:

$$
\begin{aligned}
\lim_{x \to t^-} G(x,t) &= \lim_{x \to t^+} G(x,t) \\
C_{1,1}(t) \cdot t &= C_{2,2}(t)
\end{aligned} \tag{7}
$$

$G'(x,t)$ has a jump discontinuity of $1/p(x)$, where $p(x) = 1$ is taken from the standard from of the second-or operator $L[u(x)]$:

$$
\begin{aligned}
\left.\frac{\partial G}{\partial x}\right|_{t^+} - \left.\frac{\partial G}{\partial x}\right|_{t^-} &= 1 \\
0 - C_{1,1}(t) &= 1
\end{aligned} \tag{8}
$$

From (7) and (8), we find that $C_{1,1} = -1$ and $C_{2,2}(t) = -t$. So, the completed Green's function for this operator is:

$$
G(x,t) = \begin{Bmatrix} -x, & 0 \le x \le t \\ -t, & t \le x \le 1 \end{Bmatrix} = \begin{Bmatrix} -t, & 0 \le t \le x \\ -x, & x \le t \le 1 \end{Bmatrix} \tag{9}
$$

The solution (with the change-of-variables) is then given by integrating the product of the Green's function and the forcing function:

$$
\begin{aligned}
v(x) &= \int_{x=a}^{x=b} -f(t) \quad G(x,t) \quad dt \\
&= \int_0^x -f(t)(-t)dt + \int_x^1 -f(t)(-x)dt \\
&= \frac{m\omega^2 x(-3 + x^2) + 6A\omega x \cos(\omega) - 6A \sin(\omega x)}{6\omega^2}
\end{aligned} \tag{10}
$$

Check:

$$
\begin{aligned}
\left. v(x) \right|_{x=0} &= 0 \quad \checkmark \\
\left.\frac{\partial v(x)}{\partial x}\right|_{x=1} &= 0 \quad \checkmark \\
\frac{\partial^2 v(x)}{\partial x^2} - (-f(x)) &= 0 \quad \checkmark
\end{aligned} \tag{11}
$$

The true solution can be obtained by inverting the change-of-variables:

$$\begin{aligned} u(x) &= v(x) + x/2 + 1 \\ &= 1 + \epsilon x - \frac{mx}{2} + \frac{mx^3}{6} + \frac{Ax\cos(\omega)}{\omega} - \frac{A\sin(\omega x)}{\omega^2} \end{aligned} \tag{12}$$

Check,

$$\begin{aligned} u(x)\big|_{x=0} &= 1 \quad \checkmark \\ \left.\frac{\partial u(x)}{\partial x}\right|_{x=1} &= \epsilon \quad \checkmark \\ \frac{\partial^2 u(x)}{\partial x^2} - (-f(x)) &= 0 \quad \checkmark \end{aligned} \tag{13}$$

2 Finite Element Solution

3 Results

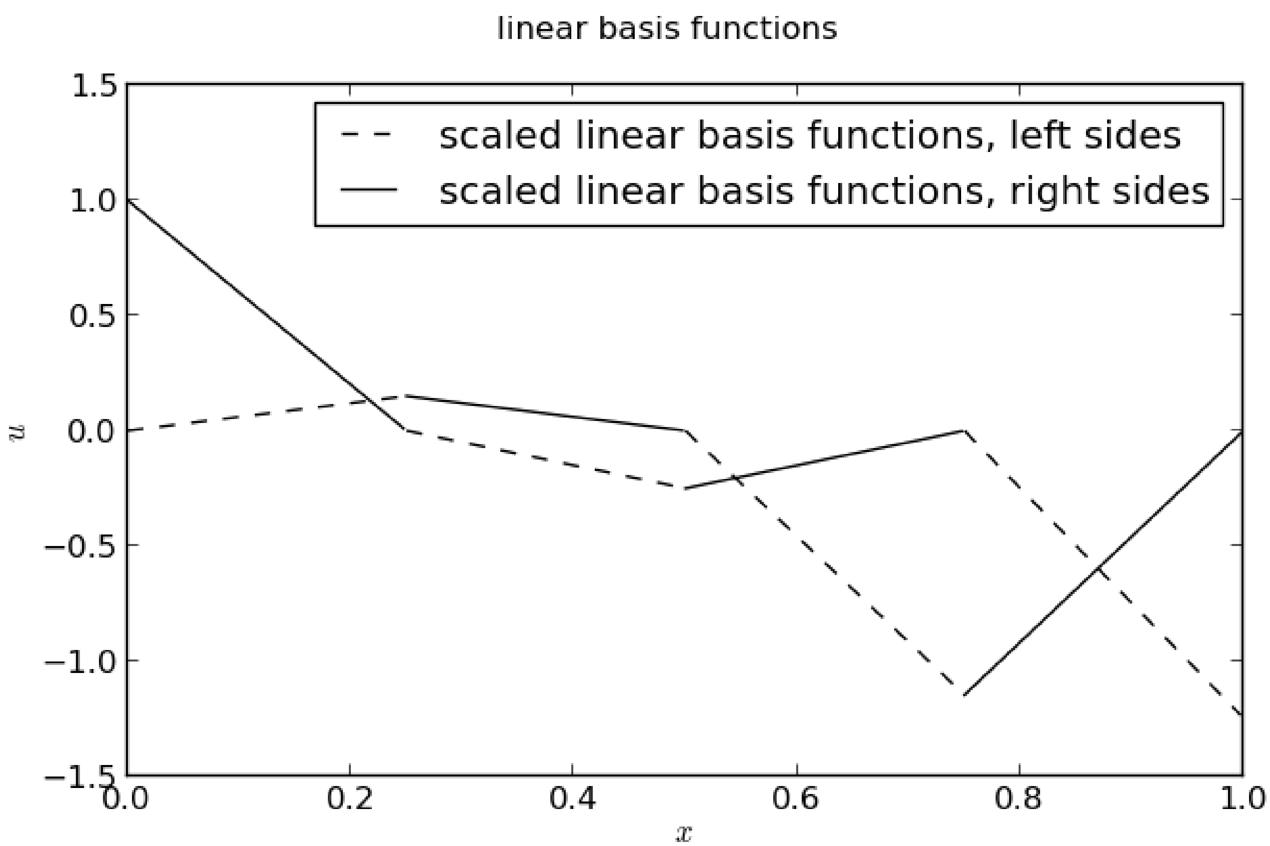

Figure 1: Five basis functions.

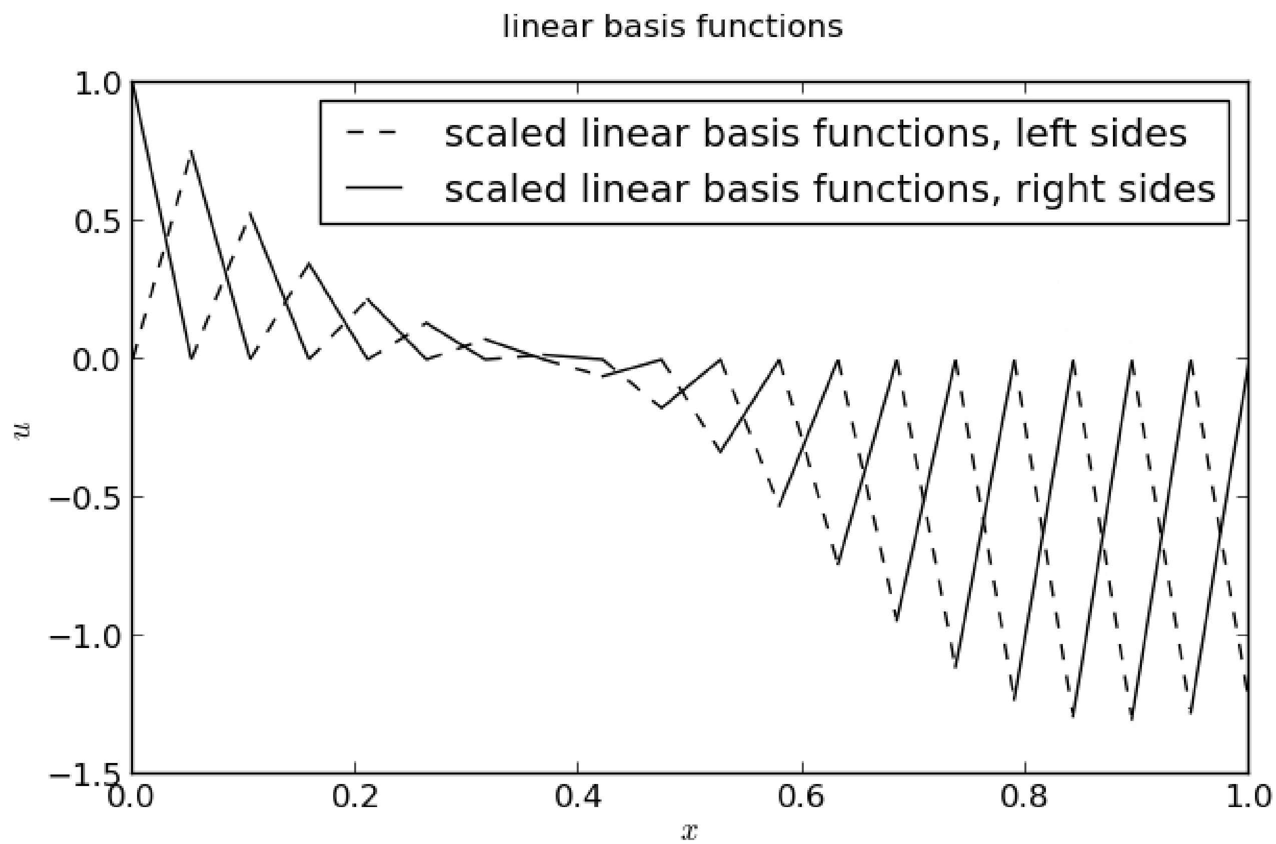

Figure 2: Twenty basis functions.

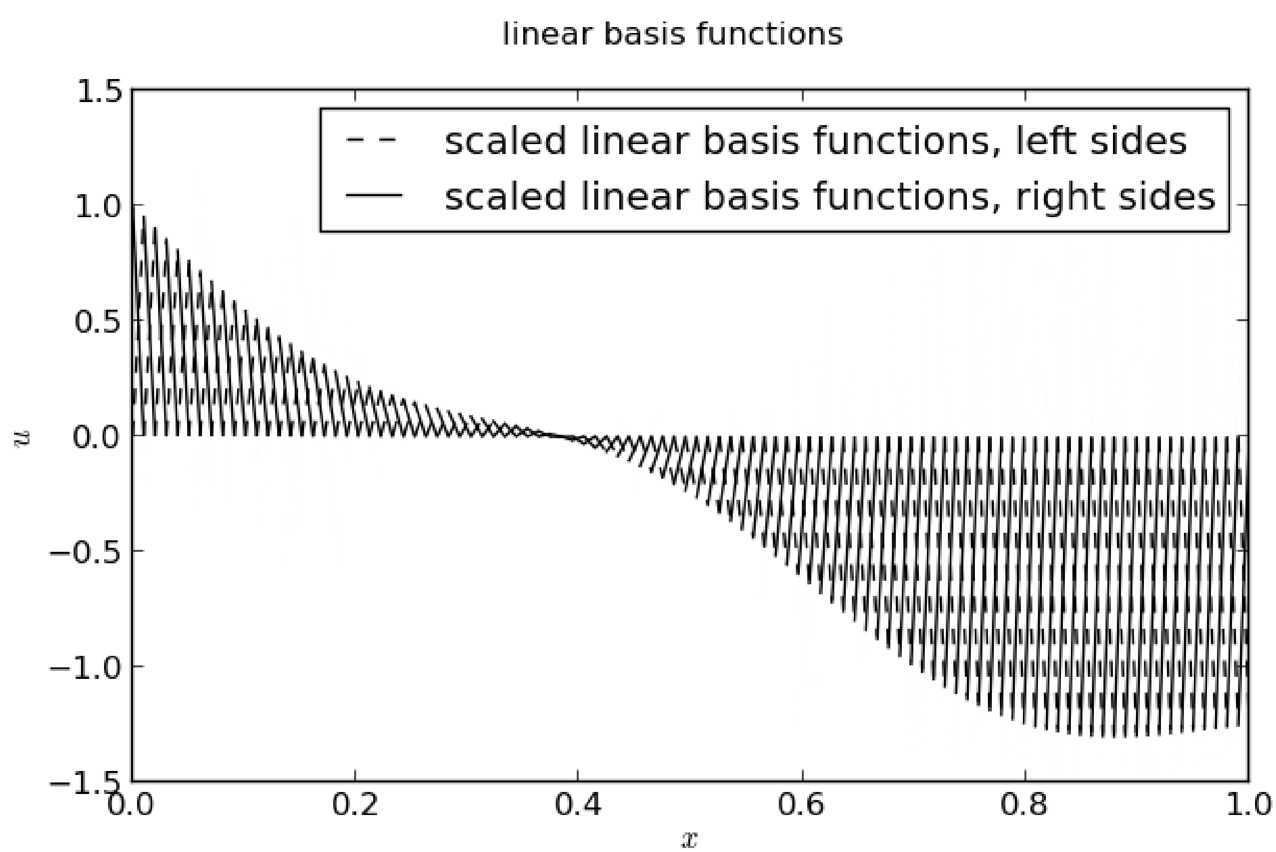

Figure 3: One hundred basis functions.

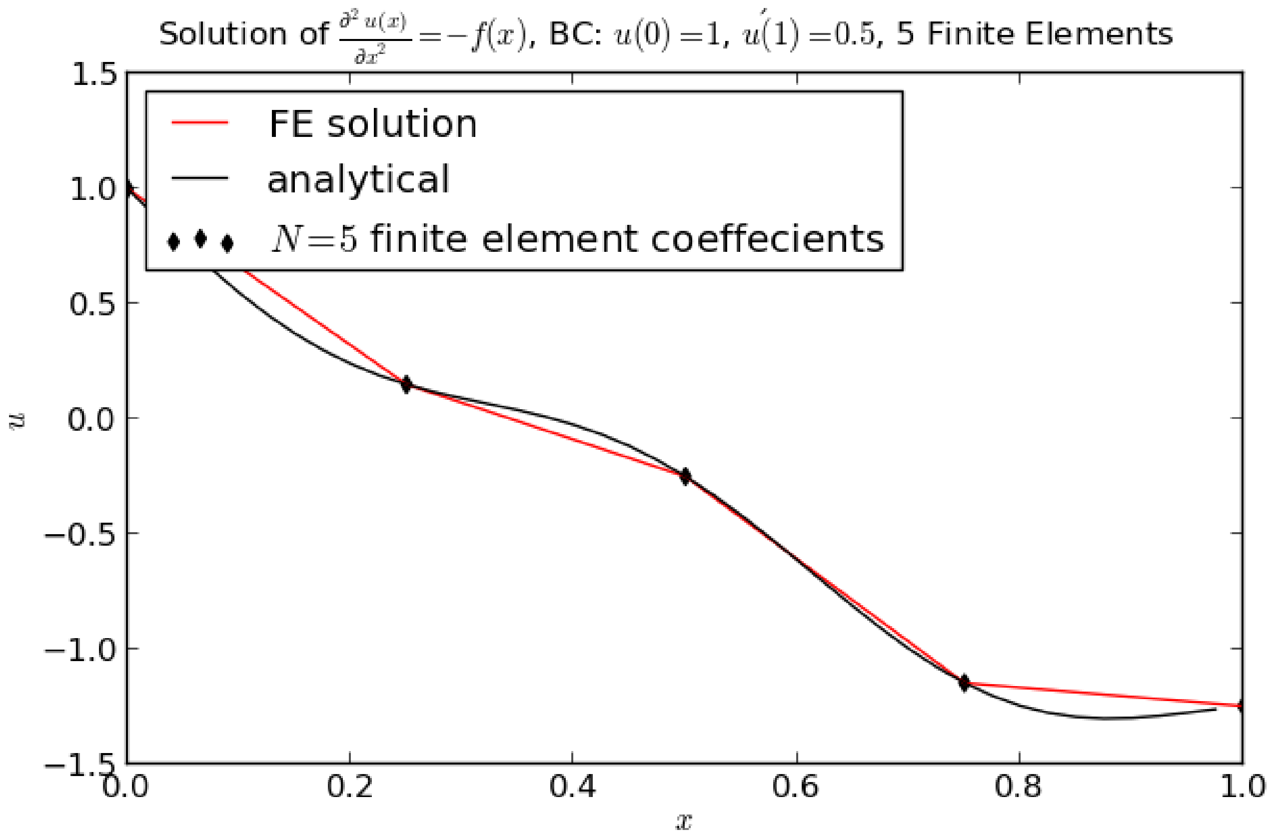

Figure 4: Solution and forcing, 5 basis functions.

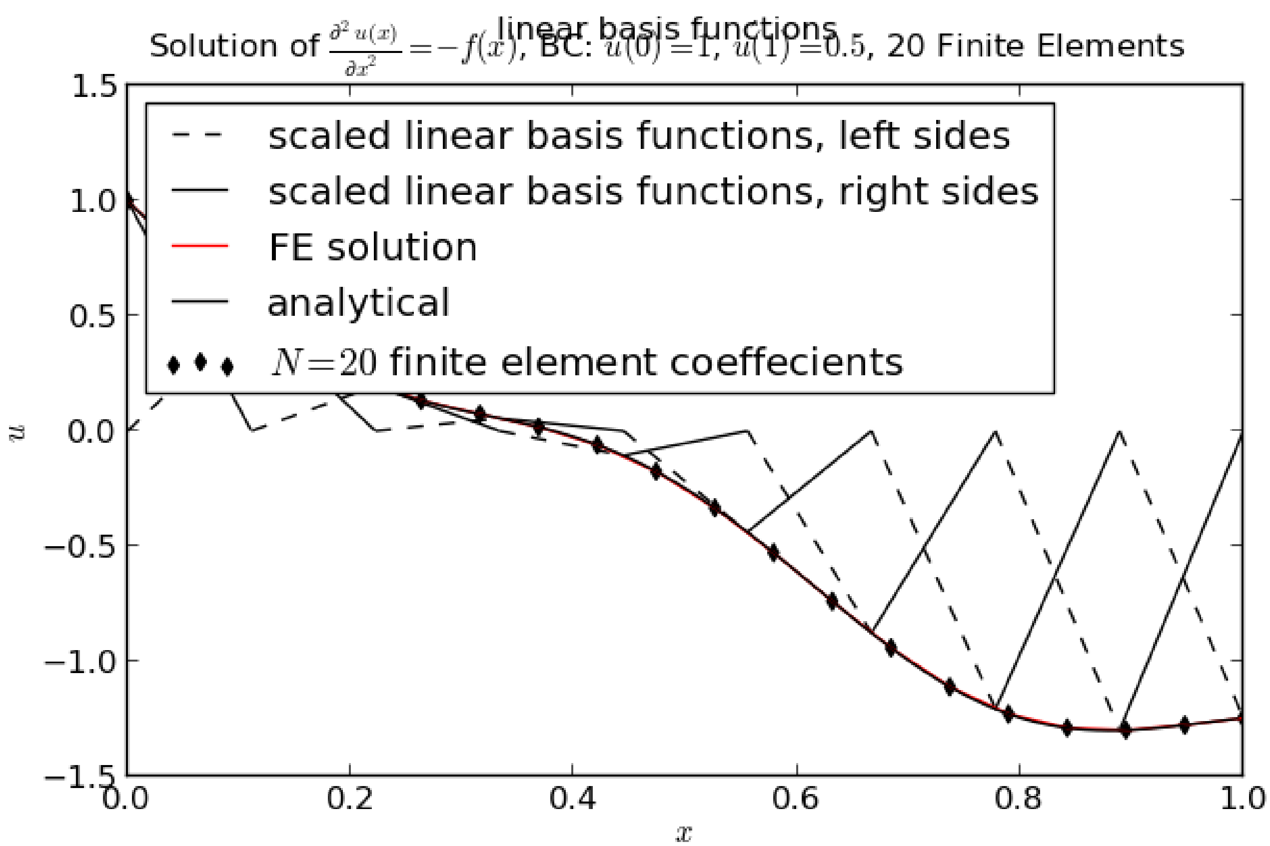

Figure 5: Solution and forcing, 20 basis functions.

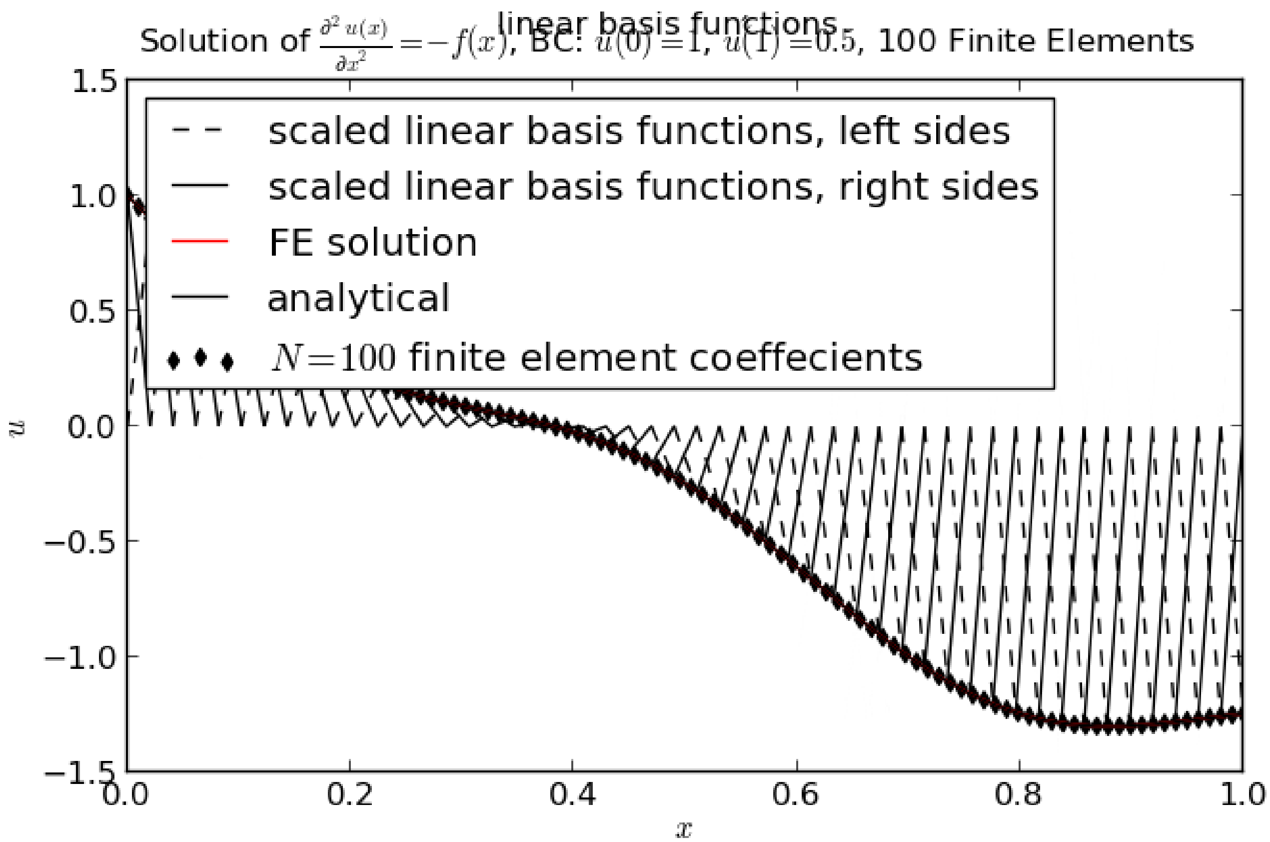

Figure 6: Solution and forcing, 100 basis functions.

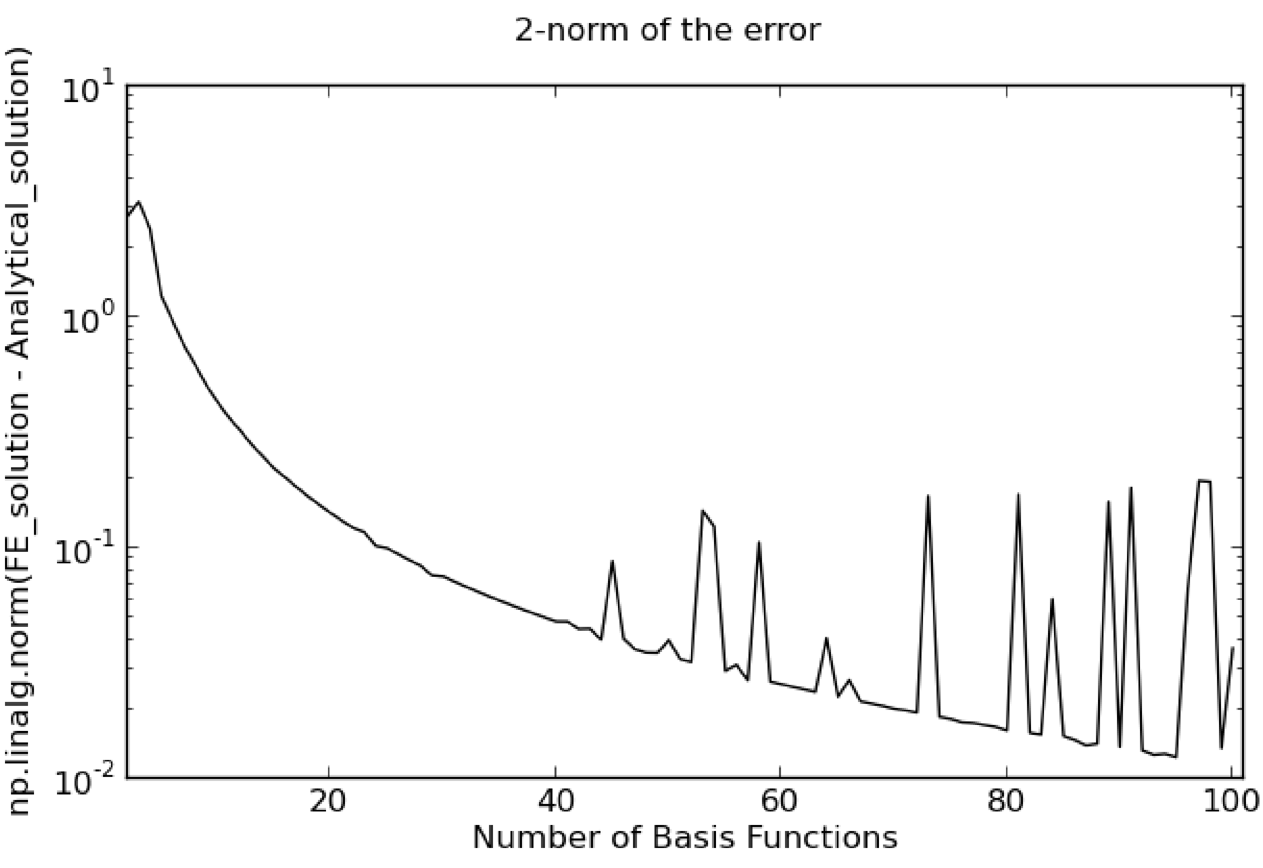

Figure 7: Rate of error reduction with increasing number of basis functions (decreasing discretization distance).

4 Code

hw7.py

```
import numpy as np
import matplotlib.pyplot as plt
width = 7.5
figsize = (width, width * 6.0 / 10.0)


def do_hw(N, save=False):

    h = (1.0 - 0) / (N - 1)

    def  phi_l(i):
        a = (i - 1) * h
        return lambda x: (x - a) / h

    def phi_r(i):
        a = (i - 1) * h
        b = a + h
        c = b + h
        return lambda x: -(x - c) / h

    cos = np.cos
    sin = np.sin

    if save:
        fig = plt.figure(figsize=figsize)
        ax = fig.add_subplot(1, 1, 1)

        fig2 = plt.figure(figsize=figsize)
        ax2 = fig2.add_subplot(1, 1, 1)
        ax2.set_xlim((0,1))
        #ax2.set_ylim((0,1))
        fig2.suptitle('linear basis functions')
        ax2.set_xlabel(r'$x$')
        ax2.set_ylabel(r'$u$')

    epsilon = 0.5
    omega = 10.0
    m = 4.0
    amp = 18.0
    forcing = lambda x: -amp * sin(x * omega) - m * x
    #forcing = lambda x: 0

    analytical = lambda x: 1 + epsilon*x - (m*x)/2 + (m*x**3)/6 + \
        (amp*x*cos(omega))/omega - amp*sin(omega*x)/omega**2

    def integrate(f, a, b, precision=1000):
        dx = (float(b) - float(a)) / float(precision)
        sum_ = 0
        for x in np.arange(a, b, dx):
            sum_ += f(x) * dx
        return sum_



    main = 2 / h * np.eye(N)
    tri = -1 / h * np.eye(N-1)
```

```
    tril = np.vstack((np.zeros((1, N)), np.hstack((tri, np.zeros((N-1, 1)) )) ))
    triu = np.vstack((np.zeros((1, N)), np.hstack((tri, np.zeros((N-1, 1)) )) )).T
    A = main + tril + triu
    A[0, 0] = 1
    A[0, 1] = 0
    A[-1, -1] = A[-1, -1] * 0.5
#    print "A:",
#    print A

    rhs = []
    for i in range(N):
        a = (i - 1) * h
        b = a + h
        c = b + h
        xl_l = list(np.arange(a, b, (b - a) / 100))
        yl_l = [phi_l(i)(x) for x in xl_l]

        xl_r = list(np.arange(b, c, (c - b) / 100))
        yl_r = [phi_r(i)(x) for x in xl_r]

#        if i==N-1:
#            ax2.plot(xl_l, yl_l, 'k:', label='linear basis functions, left sides')
#            ax2.plot(xl_r, yl_r, 'k-.', label='linear basis functions, right sides')
#        else:
#            ax2.plot(xl_l, yl_l, 'k:')
#            ax2.plot(xl_r, yl_r, 'k-.')

        f_l = lambda x: forcing(x) * phi_l(i)(x)
        f_r = lambda x: forcing(x) * phi_r(i)(x)
        if i==N-1:
            rhs.append(integrate(f_l, a, b) + epsilon)
        else:
            rhs.append(integrate(f_l, a, b) + integrate(f_r, b, c))

    rhs = np.array(rhs).reshape((N, 1))
    rhs[0] = 1
#    print "rhs:",
#    print rhs
    u = np.linalg.solve(A, rhs)

    # plot scaled basis functions:
    for i in range(N):
        a = (i - 1) * h
        b = a + h
        c = b + h
        phi_i_l = lambda x: (x - a) / h * u[i]
        xl_l = list(np.arange(a, b, (b - a) / 100))
        yl_l = [phi_i_l(x) for x in xl_l]

        phi_i_r = lambda x: -(x - c) / h * u[i]
        xl_r = list(np.arange(b, c, (c - b) / 100))
        yl_r = [phi_i_r(x) for x in xl_r]

        if i==N-1:
            ax2.plot(xl_l, yl_l, 'k--', label='scaled linear basis functions, left sides')
            ax2.plot(xl_r, yl_r, 'k-', label='scaled linear basis functions, right sides')
        else:
            ax2.plot(xl_l, yl_l, 'k--')
            ax2.plot(xl_r, yl_r, 'k-')
```

```
    #v = -u + 2
#    print "u:",
#    print u
#    print "resid:",
#    print np.dot(A, u) - rhs

    xl = list(np.arange(0, 1, h))
    xl_fine = list(np.arange(0, 1, h / 10.0))
    if len(xl) < len(u):
        xl.append(1.0)

    FE_domains = []
    FE_soln = []
    for i in range(1, N):
        # in each node, two basis functions apply.
        phi_lower = phi_r(i-1)
        phi_upper = phi_l(i)
        node_soln_function = lambda x: u[i-1] * phi_lower(x) + u[i] * phi_upper(x)
        # The node streches from a to b
        a = (i - 1) * h
        b = a + h
        node_domain = list(np.arange(a, b, (b - a) / 100))
        node_soln = [node_soln_function(x) for x in node_domain]
        # Add these to the growing solution array
        FE_soln.extend(node_soln)
        FE_domains.extend(node_domain)
    if save:
        ax.plot(FE_domains, FE_soln, 'r-', label='FE solution')

    #    ax2.plot(xl_fine, [analytical(x) for x in xl_fine], 'r-', label='analytical solution')
        ax.scatter(xl, u, label=r'$N=%i$ finite element coeffecients' % N, color='k', marker='d')
        #    ax.plot(xl_fine, [-forcing(x) for x in xl_fine], 'k--', label=r'forcing function $-f(x)=%.2f s


        ax.plot(xl_fine, [analytical(x) for x in xl_fine], 'k-', label='analytical')
        fig.suptitle(r'Solution of $\frac{\partial^2 u(x) }{ \partial x^2 }= -f(x)$, BC: $u(0)=1$, $u\'(1)=
        ax.legend(loc='upper left')
        ax2.legend(loc='upper right')

        ax.set_xlim((0, 1))
        ax.set_xlabel(r'$x$')
        ax.set_ylabel(r'$u$')


        #fig.tight_layout()
        fig.savefig('hw7-solution_and_forcing-N%i.png' % N)
        fig2.savefig('hw7-basis_functions-N%i.png' % N)
        #plt.show()

    error = []
    for (FE, a) in zip(FE_soln, [analytical(x) for x in FE_domains]):
        error.append(abs(FE - a))
    return error

if __name__=="__main__":
    fig3 = plt.figure(figsize=figsize)
    ax3 = fig3.add_subplot(1, 1, 1)
    to_save = [5, 20, 100]
    N_list = []
    norm_list = []
```

```
    Nmin = 2
    Nmax = 101
    for N in range(Nmin, Nmax):
        print 'N is', N
        if N in to_save:
            error = do_hw(N, save=True)
        else:
            error = do_hw(N, save=False)
        print "  norm is ", np.linalg.norm(np.array(error))
        N_list.append(N)
        norm_list.append(np.linalg.norm(error))
#         ax3.plot(error, 'k')
    ax3.set_yscale('log')
    fig3.suptitle('2-norm of the error')
    ax3.set_xlim((Nmin, Nmax))
    ax3.set_xlabel('Number of Basis Functions')
    ax3.set_ylabel('np.linalg.norm(FE_solution - Analytical_solution)')
    ax3.plot(N_list, norm_list, 'k')
    fig3.savefig('hw7-error_rate.png')
```