CBE 502 - HW7 - Finite Elements

Tom Bertalan

December 26, 2012

Code to make this document is online at `github.com/tsbertalan-homework/502hw7`.

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:

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

Use a change of variables to create homogeneous boundary conditions.

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

To make the new boundary conditions homogeneous (Dirichlet at $x = 0$ and Neumann at $x = 1$), 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 Solution by Properties of the Green's Function

$G(x,t)$ satisfies the homogeneous problem (that is, $\partial^2 G(x,t)/\partial x^2 = 0$):

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

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

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

$G(x,t)$ 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-order operator $L[v(x)]$:

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

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 L 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:

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

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

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

Check,

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

Algebra is verified in Section 5.

2 Finite Element Solution

Galerkin Form

$$L[\tilde{u}(x)] - f(x) = error(x) \approx \vec{0}(x) \tag{14}$$

$$\tilde{u}(x) = \sum_{i=1}^{N} u_i \phi^i(x) \tag{15}$$

$$\int_a^b \left[L[\tilde{u}(x)] - f(x)\right] \phi^i(x) dx = 0 \tag{16}$$

Split this into two separate integrals, then in the left integral, let $\mu = \phi^i(x)$ and $d\eta = \frac{\partial^2 \tilde{u}}{\partial x^2} dx$. Integrate by parts ($\int \mu d\eta = \mu\eta - \int \eta d\mu$). This reduces the derivatives in the operator from second to first order, which enables the use of linear basis functions.

$$\begin{aligned} \int_a^b L[\tilde{u}(x)]\phi^i(x)dx - \int_a^b f(x)\phi^i(x)dx = 0 \\ \int_0^1 \frac{\partial^2 \tilde{u}}{\partial x^2}\phi^i dx - \int_0^1 f(x)\phi^i(x)dx = 0 \end{aligned} \tag{17}$$

$$\begin{aligned} \phi^i(x)\left.\frac{\partial \tilde{u}}{\partial x}\right|_0^1 - \int_0^1 \left[\frac{\partial \tilde{u}}{\partial x}\frac{\partial \phi^i}{\partial x} + f(x)\phi^i(x)\right] \\ \phi^i(x)\left.\frac{\partial \tilde{u}}{\partial x}\right|_0^1 - \int_0^1 \frac{\partial \tilde{u}}{\partial x} = \\ \frac{\partial \phi^i}{\partial x}dx = \int_0^1 f(x)\phi^i(x) \end{aligned} \tag{18}$$

Now use (15) to express $\partial\tilde{u}/\partial x$ in terms of basis functions. Bring the derivative inside the sum:

$$\phi^i(x)\sum u_j \frac{\partial \phi^i}{\partial x} - \int_0^1 \frac{\partial \phi^i}{\partial x}\left(\sum u_j \frac{\partial \phi^j}{\partial x}\right) dx = \int_0^1 f(x)\phi^i(x)dx \tag{19}$$

Since all other basis functions are zero at the right side of the domain, the term on the left applies only to the last basis function. This allows the use of a Dirac delta function. Combined with the right (Neumann) boundary condition, this changes the term to $\epsilon\delta_{iN}$. The derivative of ϕ^i can be brought inside the adjacent sum. Its product with the derivative of ϕ^j is zero for values of j but $j = i-1$, $j = i$, and $j = i+1$, by the near-orthogonality of the basis functions. With our linear basis functions, the right-hand-side intergrals must be calculated in piecewise fashion, with the first piece being from the left edge of each hat function to its point, and the second piece being from the point to the right edge.

This simplifies the equation somewhat.

$$\epsilon\delta_{iN} - \int_0^1 \frac{\partial\phi^i}{\partial x}\left(\sum_{j=i-1}^{i+1} u_j \frac{\partial\phi^j}{\partial x}\right) dx = \int_0^1 f(x)\phi^i(x)dx$$
$$\sum_{j=1}^{N}\left(\int_0^1 \frac{\partial\phi^i}{\partial x}\frac{\partial\phi^j}{\partial x}dx\right) u_j = \epsilon\delta_{iN} - \int_0^1 f(x)\phi^i(x)dx \tag{20}$$
$$\underline{\underline{K}} \cdot \underline{u} = \underline{F}$$

Here, $\underline{\underline{K}}$ is a tridiagonal square matrix. To satisfy the left boundary condition, the first row of $\underline{\underline{K}}$ is $1, 0, 0, ...$, and the first element of $\underline{F}$ is 1.

With linear basis functions, most of the main diagonal of $\underline{\underline{K}}$ (besides the first element) is repetitions of $2/h$. Here, $h = (1-0)/(N-1)$ is half the width of one of the N basis functions. However, the last value in the main diagonal is only $1/h$, because the product $\frac{\partial\phi^N}{\partial x}\frac{\partial\phi^N}{\partial x}$ only involves the left slope of the hat for the final basis function. Besides the second element in the first row, which is zero to satisfy the left boundary condition, the two off-diagonals are repetitions of $-1/h$, from the product of the (opposite-sign) slopes of the overlapping portions of adjacent basis functions.

In Section 3, we solve this system with the python library *Numpy* ("numerical python") for various numbers of basis functions, N.

3 Results

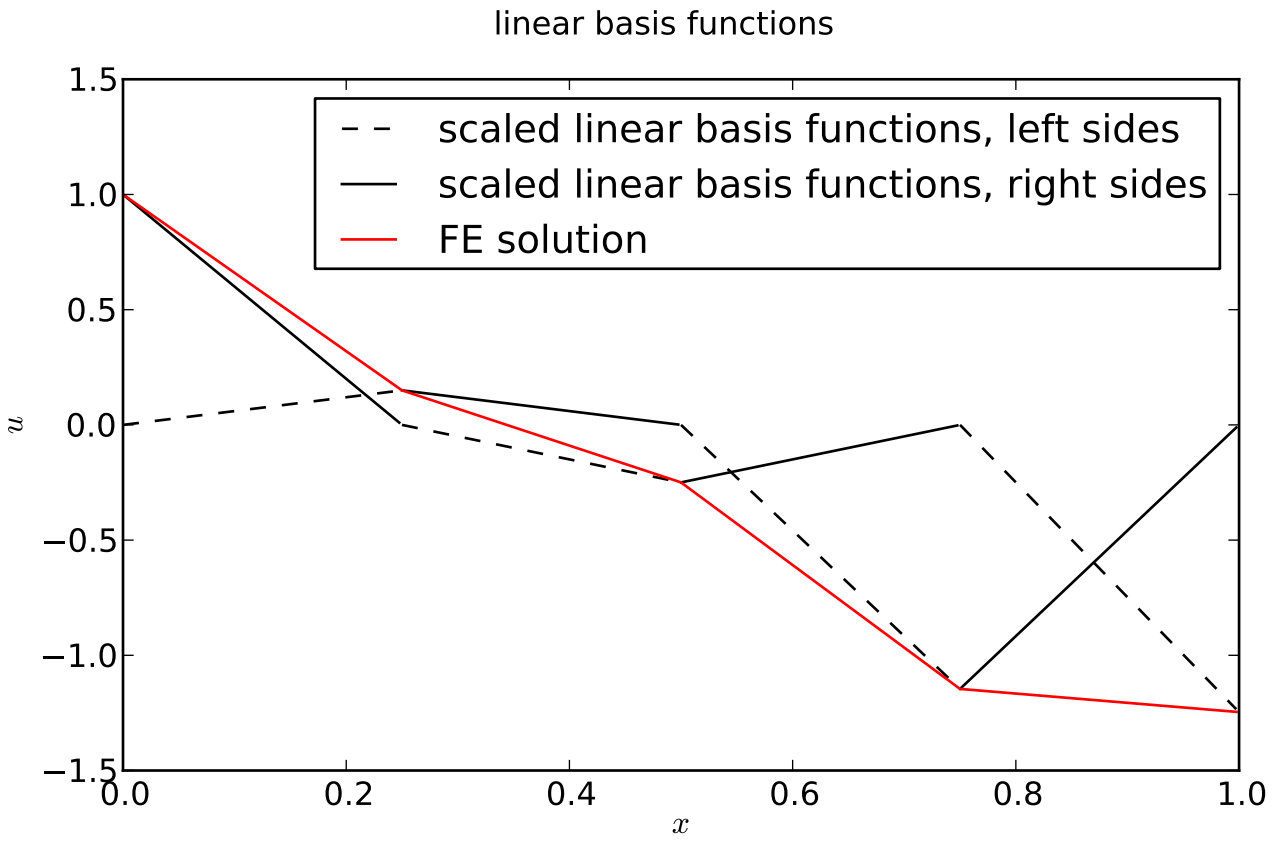

Figure 1: Five basis functions. Unscaled basis functions are hats of width $2h$ and unit height. The FE (finite element) solution is the superposition of the scaled basis functions shown here: $u(x) = \sum_{j=1}^{N} u_j\phi^j(x)$

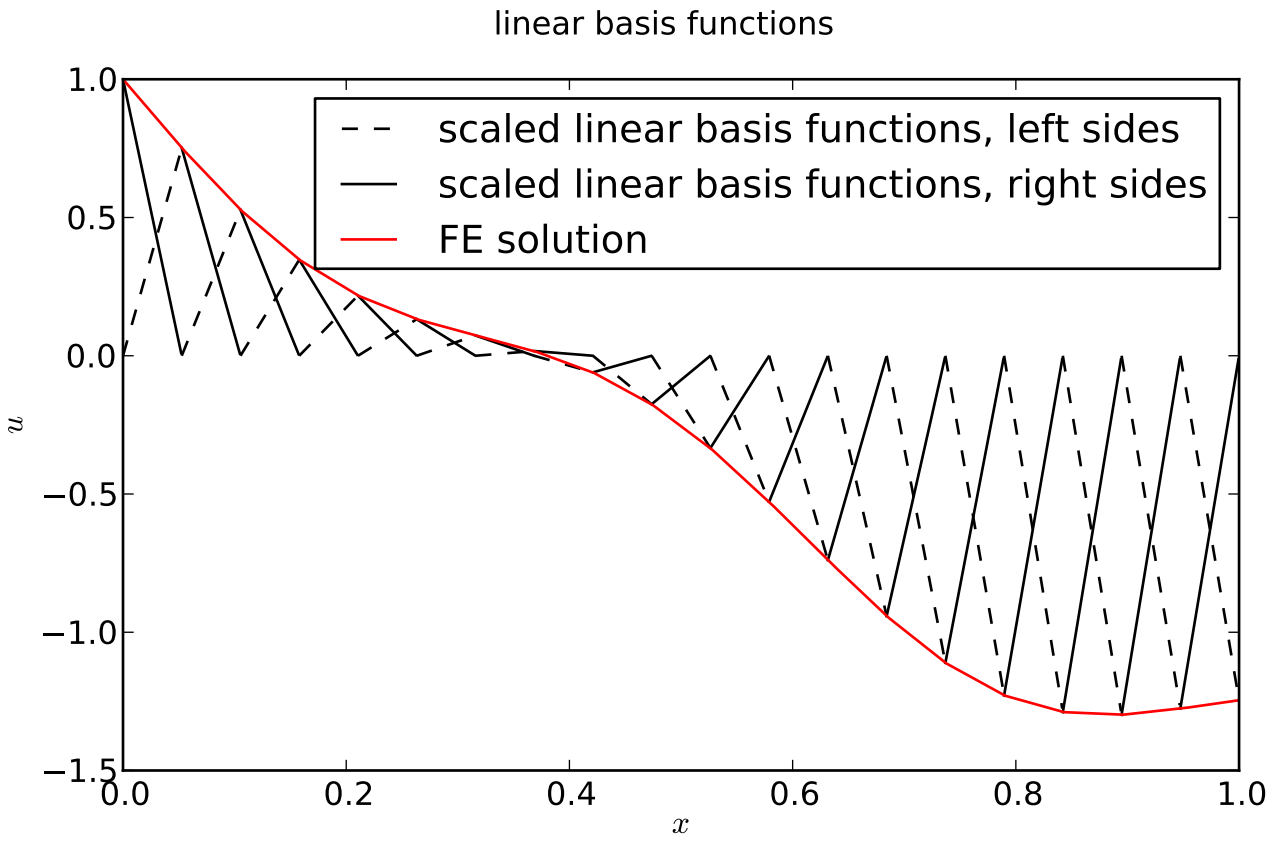

Figure 2: Twenty basis functions.

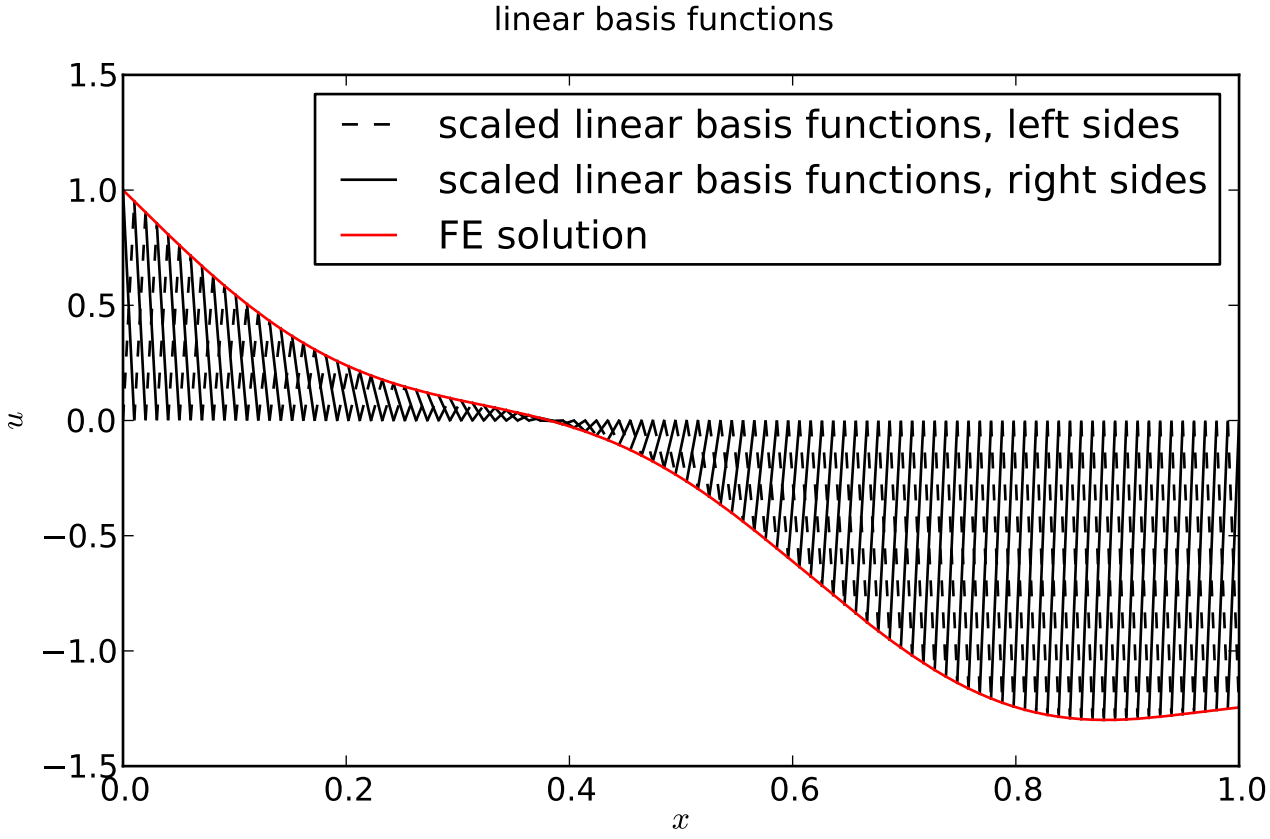

Figure 3: One hundred basis functions.

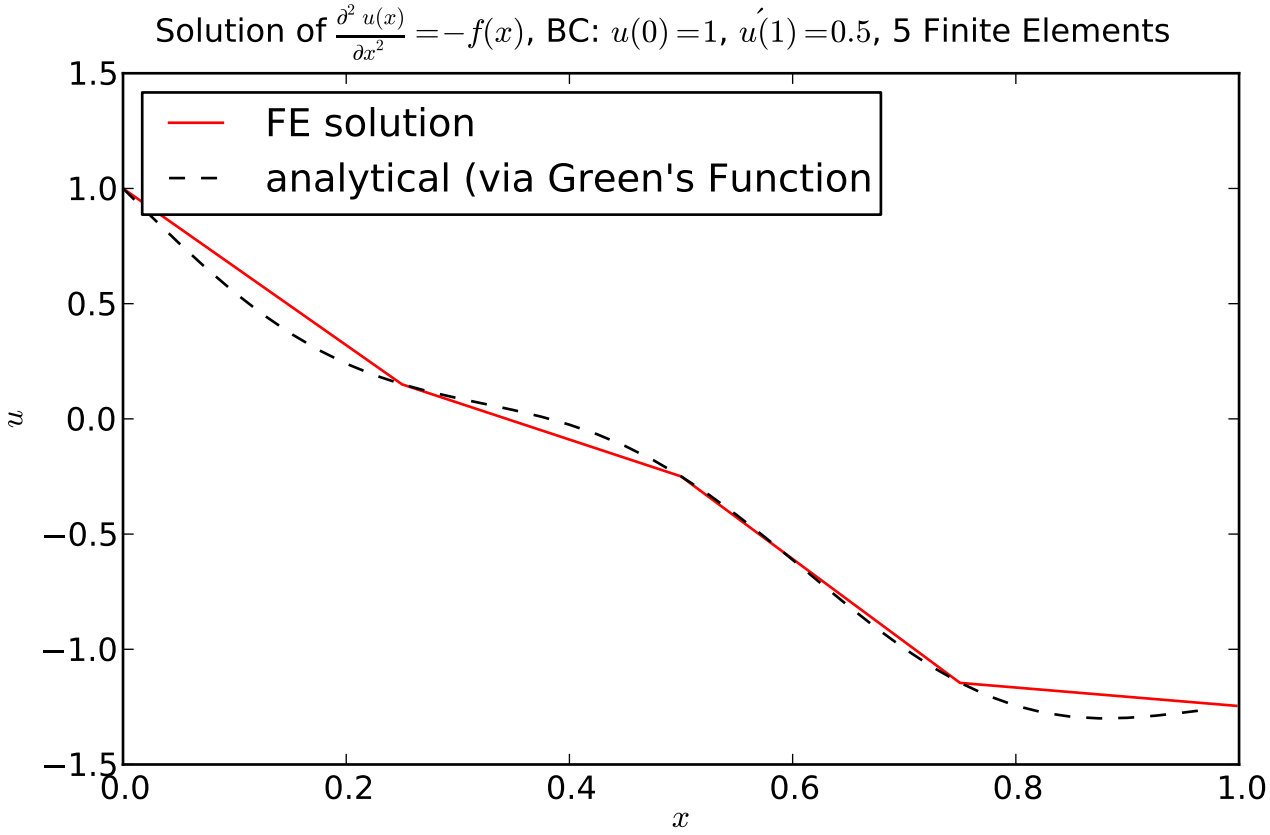

Figure 4: Solution and forcing, 5 basis functions.

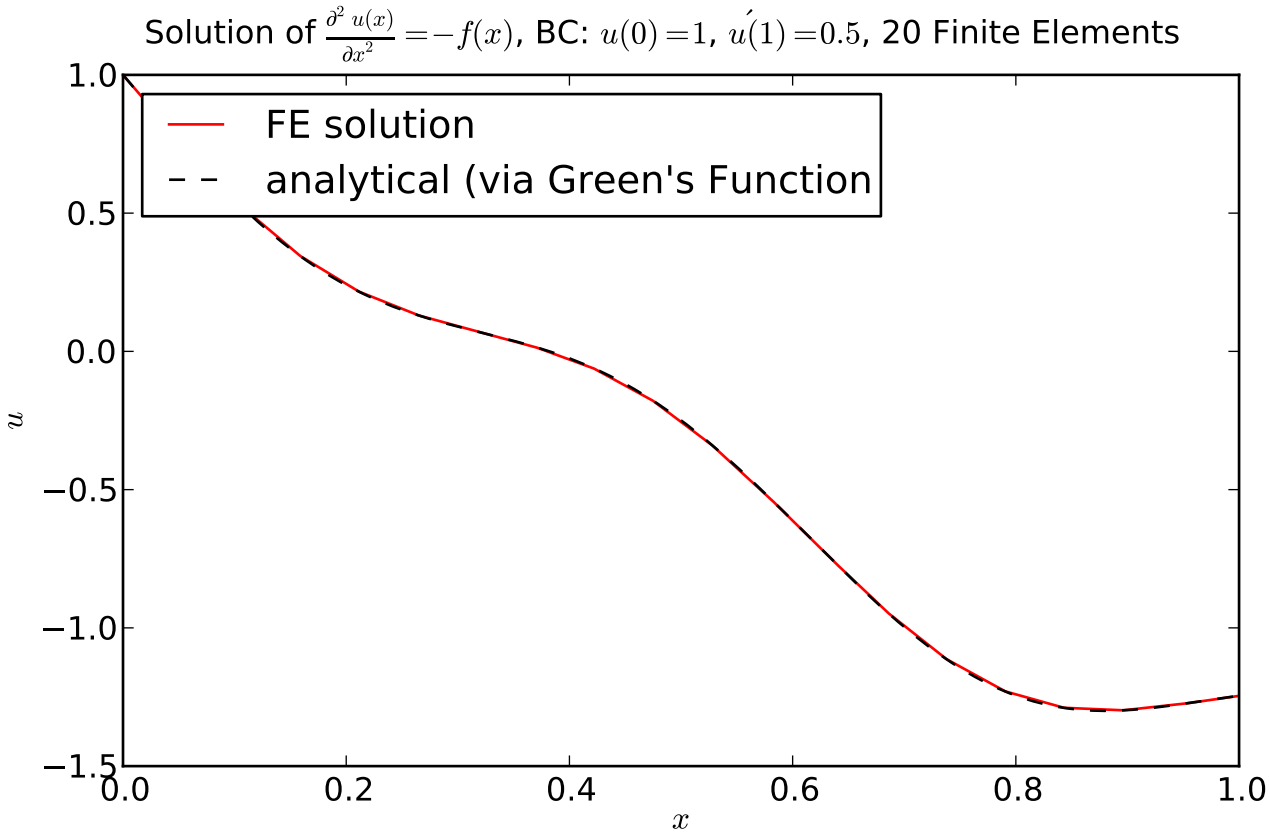

Figure 5: Solution and forcing, 20 basis functions.

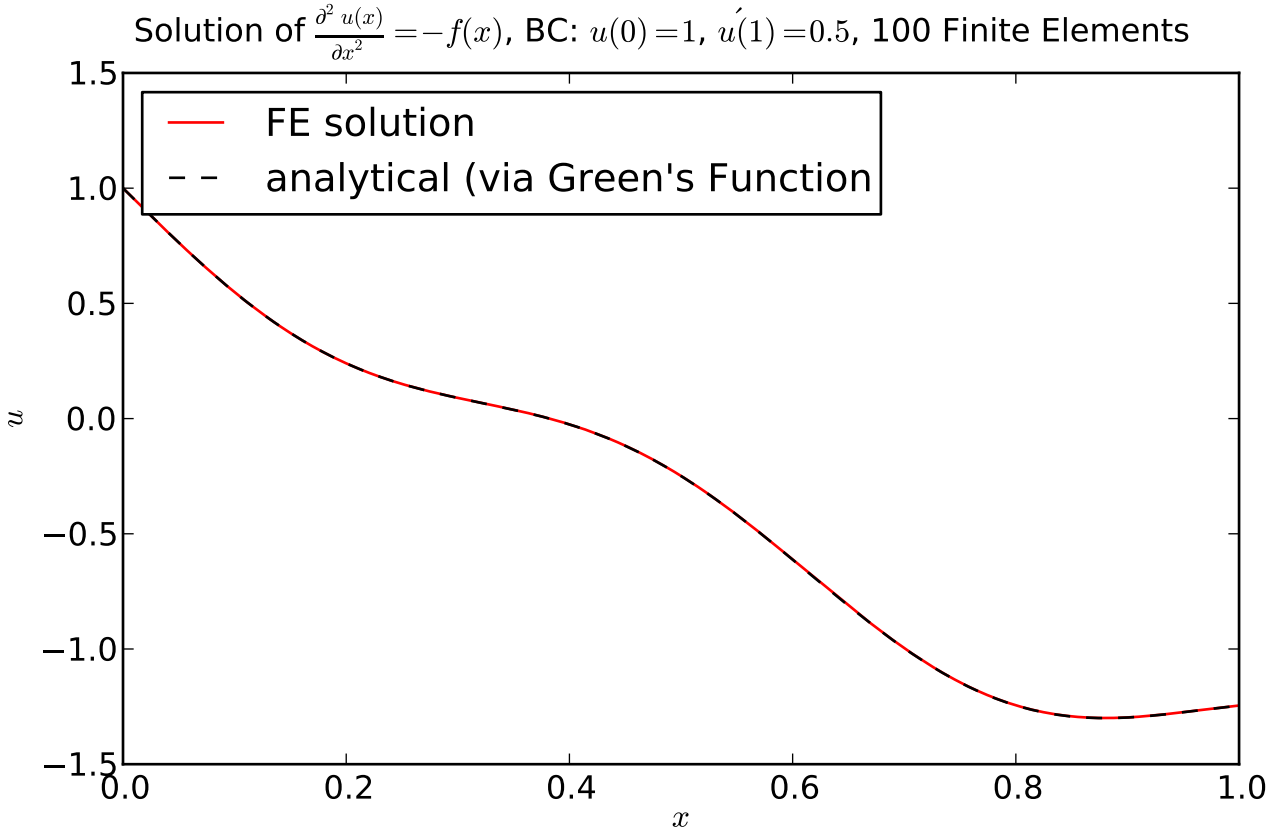

Figure 6: Solution and forcing, 100 basis functions.

Order	Vector Norm
None	2-norm, $[\sum_i abs(a_i)^2]^{1/2}$
`inf`	`max(abs(x))`
`-inf`	`min(abs(x))`
0	`sum(x != 0)`
1	as below
-1	as below
2	as below
-2	as below
other	`sum(abs(x)**ord)**(1./ord)`

Table 1: To compare the finite element and Green's function solutions, we need a vector norm to operate on the difference of their solution vectors. Nine matrix norms are available in `numpy.linalg.norm`. However, all appear to give the same behavior (see Figure 7). `inf` is the `numpy.inf` object.

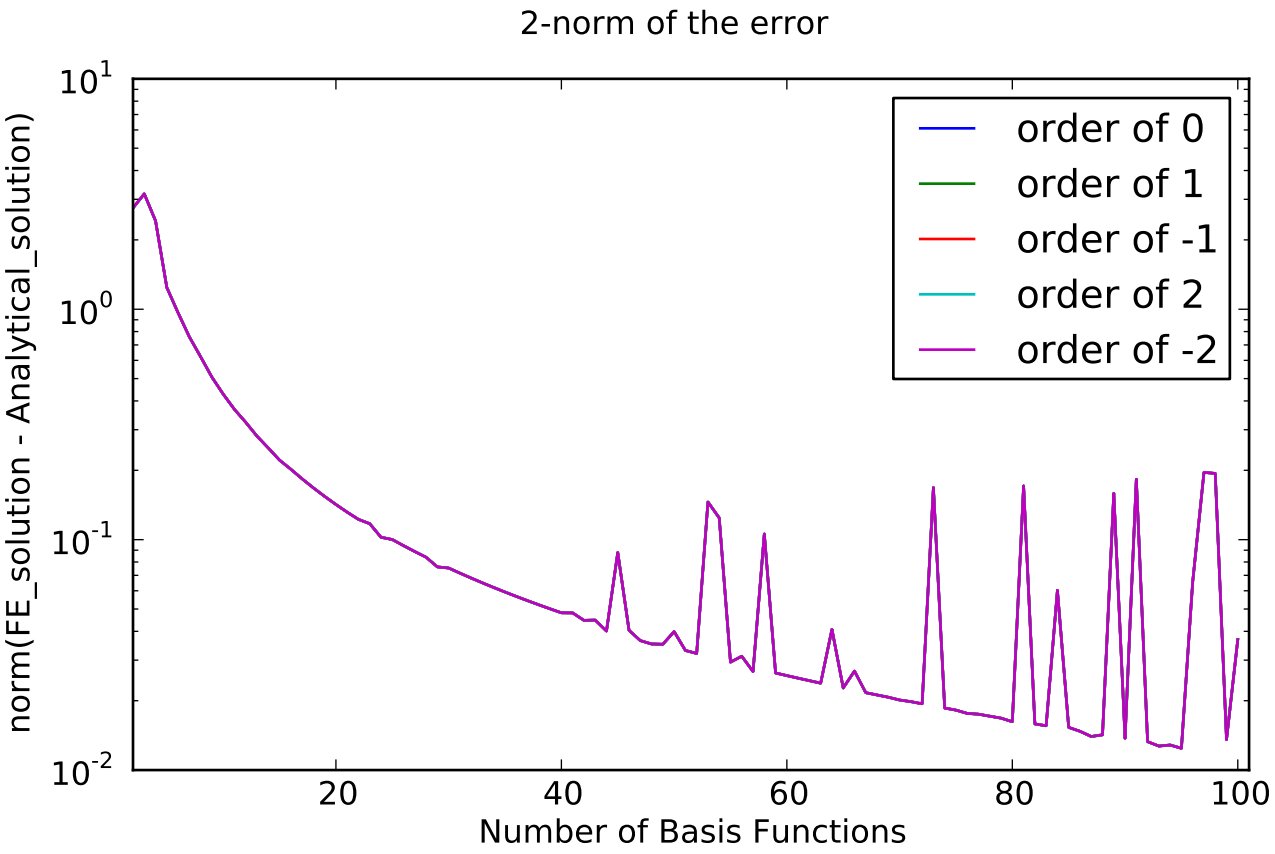

Figure 7: Rate of error reduction with increasing number of basis functions (decreasing discretization distance).

4 Code

hw7.py

```
from numpy import cos, sin, eye, arange, vstack, hstack, zeros, array
from numpy.linalg import norm, solve
import matplotlib.pyplot as plt
figWidth = 7.5
figsize = (figWidth, figWidth * 6.0 / 10.0)


def do_hw(N, save=False):

    # Set up figures.
    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'$u$')

    h = (1.0 - 0) / (N - 1)
    # Define the left and right pieces of the basis functions as separate linear functions.
    # These are functions that return functions.

    af = lambda i: (i - 1) * h

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

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

    def integrate(f, a, b, precision=1000):
        '''Simple Reimann-sum integrator. Could be improved by using
        trapezoidal integration. Probably doesn't perform well for high-slope
        functions (f).'''
        dx = (float(b) - float(a)) / float(precision)
        sum_ = 0
        for x in arange(a, b, dx):
            sum_ += f(x) * dx
        return sum_

    # Set up problem.
    epsilon = 0.5
    omega = 10.0
    m = 4.0
    amp = 18.0
    forcing = lambda x: -amp * sin(x * omega) - m * x

    analytical = lambda x: 1 + epsilon*x - (m*x)/2 + (m*x**3)/6 + \
```

```
    (amp*x*cos(omega))/omega - amp*sin(omega*x)/omega**2

# Construct the coeffecient matrix.

main = 2 / h * eye(N)
tri = -1 / h * eye(N-1)
tril = vstack((zeros((1, N)), hstack((tri, zeros((N-1, 1)) )) ))
triu = vstack((zeros((1, N)), hstack((tri, zeros((N-1, 1)) )) )).T
A = main + tril + triu

# Include boundary conditions' modifications.
A[0, 0] = 1
A[0, 1] = 0
A[-1, -1] = A[-1, -1] * 0.5

# Construct the right-hand-side column vector.
rhs = []
for i in range(N):
    a = af(i)  # x=a at the left edge of the ith basis function,
    b = a + h  # b at the point,
    c = b + h  # & c at the right edge.

    # Create functions to be integrated for the right-hand-side vector.
    f_l = lambda x: forcing(x) * phi_l(i)(x)
    f_r = lambda x: forcing(x) * phi_r(i)(x)
    if i==N-1:  # right (Neumann) boundary condition
        rhs.append(integrate(f_l, a, b) + epsilon)
    else:
        rhs.append(integrate(f_l, a, b) + integrate(f_r, b, c))

rhs = array(rhs).reshape((N, 1))
rhs[0] = 1  # left (Dirichlet) boundary condition

# Solve for the coeffecients of the basis functions.
# Another option here is github.com/tsbertalan/openmg
u = solve(A, rhs)

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

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

    if save:
        if i==N-1:  # We only want one label in the plot legend, not N labels.
            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-')

# Compose the finite element solution as the sum of the product of
# coeffecients u[j] and basis functions phi_l(i) and phi_r(i).
```

```
    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 distance of length h.
        a = af(i)
        b = a + h
        node_domain = list(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:
        xl = list(arange(0, 1, h))
        xl_fine = list(arange(0, 1, h / 10.0))  # For plotting the analytical solution.
        if len(xl) < len(u):  # For very small N (~4), we sometimes lose x=1.0
            xl.append(1.0)
        ax2.plot(FE_domains, FE_soln, 'r-', label='FE solution')
        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 sin(

        ax.plot(xl_fine, [analytical(x) for x in xl_fine], 'k--', label="analytical (via Green's Function")
        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.savefig('hw7-solution_and_forcing-N%i.pdf' % N)
        fig2.savefig('hw7-basis_functions-N%i.pdf' % N)

    # The error is the vector difference of the finite element solution and
    # the Green's function solution, at the same x points.
    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__":
    # For each of several choices of norm, plot the decreasing norm of
    # the error as the number of basis functions increases.
    fig3 = plt.figure(figsize=figsize)
    ax3 = fig3.add_subplot(1, 1, 1)
    to_save = [5, 20, 100]
    N_list = []
    Nmin = 2
    Nmax = 101
    orders=[0, 1, -1, 2, -2]
    norm_list = [[] for order in orders]
    ordernames=['order of %i' % order for order in orders]
    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)
    N_list.append(N)
    for normIndex in range(len(orders)):
        norm_list[normIndex].append(norm(error, ord=order))
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('norm(FE_solution - Analytical_solution)')
for normIndex in range(len(orders)):
    ax3.plot(N_list, norm_list[normIndex], label=ordernames[normIndex])
ax3.legend()
fig3.savefig('hw7-error_rate-mult_orders.pdf')
```

5 Algebra by *Mathematica*

5.1 Generalized in A, ω, m, ϵ

```
In[1]:= (*omega := 10.0;*)
        (*m := 4.0;*)
        (*epsilon := 0.5;*)
        (*amp = 18.0*)
        f[x_] := -amp * Sin[omega * x] - m * x
        oldf[x_] := -4 * x

In[3]:= v[x_] :=  Integrate[ -f[t]*(-t), {t, 0, x} ] + Integrate[ -f[t]*(-x), {t, x, 1} ];
        Simplify[v[x]]

Out[4]= (m omega^2 x (-3 + x^2) + 6 amp omega x Cos[omega] - 6 amp Sin[omega x]) / (6 omega^2)

In[5]:= u[x_] := v[x] + epsilon * x + 1

In[6]:= Simplify[u[x]]

Out[6]= 1 + epsilon x - m x/2 + m x^3/6 + amp x Cos[omega]/omega - amp Sin[omega x]/omega^2

In[7]:= Simplify[D[u[x], x]]

Out[7]= epsilon - m/2 + m x^2/2 + amp Cos[omega]/omega - amp Cos[omega x]/omega

In[8]:= v[0] // N
        D[v[x], x] /. x → 1 // N
        Simplify[D[D[v[x], x] + f[x]]

Out[8]= 0.

Out[9]= -1. m - 1. amp Sin[omega] + 0.5 (2. m omega + 2. amp omega Sin[omega]) / omega

Out[10]= 0

In[11]:= u[0]
         D[u[x], x] /. x → 1
         Simplify[D[D[u[x], x] + f[x]]

Out[11]= 1

Out[12]= epsilon - m - amp Sin[omega] + (2 m omega + 2 amp omega Sin[omega]) / (2 omega)

Out[13]= 0
```

5.2 Particularized, to Show Satisfied BCs and Nonhomogeneous Equation

```
In[14]:= omega := 10.0;
         m := 4.0;
         epsilon := 0.5;
         amp = 18.0
         f[x_] := -amp * Sin[omega * x] - m * x
         oldf[x_] := -4 * x
Out[17]= 18.

In[20]:= v[x_] :=  Integrate[ -f[t]*(-t), {t, 0, x} ] + Integrate[ -f[t]*(-x), {t, x, 1} ];
         Simplify[v[x]]
Out[21]= 0. - 3.51033 x + 0.666667 x^3 - 0.18 Sin[10. x]

In[22]:= u[x_] := v[x] + epsilon * x + 1

In[23]:= Simplify[u[x]]

Out[23]= 1. - 3.01033 x + 0.666667 x^3 - 0.18 Sin[10. x]

In[24]:= Simplify[D[u[x], x]]

Out[24]= -3.01033 + 2. x^2 - 1.8 Cos[10. x]

In[25]:= v[0] // N
         D[v[x], x] /. x → 1 // N
         Simplify[D[D[v[x], x], x] + f[x]]
Out[25]= 0.

Out[26]= -8.88178 × 10^-16

Out[27]= 0. - 3.55271 × 10^-15 Sin[10. x]

In[28]:= u[0]
         D[u[x], x] /. x → 1
         Simplify[D[D[u[x], x], x] + f[x]]
Out[28]= 1.

Out[29]= 0.5

Out[30]= 0. - 3.55271 × 10^-15 Sin[10. x]
```