Homework 4

Computational Math

Author:
Mikola Lysenko

May 13, 2010

1

a Let u be a solution to the given ODE. Then, for any 2π periodic smooth function v it must be that:

$$\int_{-\pi}^{\pi} \left(u_{xx}(x) + \left(\sin(x) + \cos(x) + 2\sin(x)\cos(x)\right) u(x)\right) v(x) dx = \int_{-\pi}^{\pi} e^{\sin(x)} e^{\cos(x)} v(x) dx$$

By linearity, we can split the righthand side into homogeneous components and apply integration by parts. For the second order components we get:

$$\begin{aligned} \int_{-\pi}^{\pi} u_{xx}(x) v(x) dx &= u_x(x) v(x)|_{-\pi}^{\pi} - \int_{-\pi}^{\pi} u_x(x) v_x(x) dx \\ &= -\int_{-\pi}^{\pi} u_x(x) v_x(x) dx \\ &= p_2(u, v) \end{aligned}$$

For the 0th order terms, no additional work is necessary to get a bilinear form:

$$\int_{-\pi}^{\pi} \left(\sin(x) + \cos(x) + 2\sin(x)\cos(x)\right) u(x) v(x) dx = p_0(x)$$

And finally for the right hand side, we construct $f(v)$:

$$\int_{-\pi}^{\pi} e^{\sin(x)} e^{\cos(x)} v(x) dx = f(v)$$

By construction our space of test functions satisifies the boundary, so we have the following weak form for the system:

$$u^T (p_2 + p_0) v = f^T v$$

b Fix a collection of n ordered grid points $X = \{x_i \in [-\pi, \pi) | x_i < x_{i+1}, i \in [0, N)\}$. To get a cG(1) finite element discretization, we restrict the space of test functions and solutions to piecewise linear maps over this grid, which is spanned by the basis of hat functions. We define the i^{th} basis hat function, $\varphi^i(x)$, as follows:

$$\varphi^i(x) = \begin{cases} \frac{x_{i+1} - x}{x_{i+1} - x_i} & \text{if } x \in [x_i, x_{i+1}) \\ \frac{x - x_{i-1}}{x_i - x_{i-1}} & \text{if } x \in [x_{i-1}, x_i) \\ 0 & \text{otherwise} \end{cases}$$

Boundary conditions are applied to make the test functions cyclic. This may be done by just taking the indexes mod n and the difference terms mod 2π. Substituting this into the above equation gives the following matrix equation for the discrete Galerkin equation:

$$p_2(\varphi^i, \varphi^j) + p_0(\varphi^i, \varphi^j) = f(\varphi^j)$$

Now we expand each term separately and integrate piecewise. Starting with p_2:

$$
\begin{aligned}
p_2(\varphi^i, \varphi^j) &= -\int_{-\pi}^{\pi} \varphi_x^i(x)\varphi_x^j(x)dx \\
&= (\delta_{i,j+1} - \delta_{i,j}) \int_{x_{i-1}}^{x_i} \left(\frac{1}{x_i - x_{i-1}}\right)^2 dx + (\delta_{i,j-1} - \delta_{i,j}) \int_{x_i}^{x_{i+1}} \left(\frac{1}{x_{i+1} - x_i}\right)^2 dx \\
&= (\delta_{i,j+1} - \delta_{i,j}) \frac{1}{x_i - x_{i-1}} + (\delta_{i,j-1} - \delta_{i,j}) \frac{1}{x_{i+1} - x_i}
\end{aligned}
$$

Next we deal with p_0:

$$
\begin{aligned}
p_0(\varphi^i, \varphi^j) &= \int_{-\pi}^{\pi} (\sin(x) + \cos(x) + 2\sin(x)\cos(x))\, \varphi^i(x)\varphi^j(x)dx \\
&= \int_{x_{i-1}}^{x_i} (\sin(x) + \cos(x) + 2\sin(x)\cos(x)) \frac{(x - x_{i-1})(\delta_{i,j}(x - x_{i-1}) + \delta_{i,j+1}(x_i - x))}{(x_i - x_{i-1})^2} dx + \\
&\quad \int_{x_i}^{x_{i+1}} (\sin(x) + \cos(x) + 2\sin(x)\cos(x)) \frac{(x_{i+1} - x)(\delta_{i,j}(x_{i+1} - x) + \delta_{i,j-1}(x - x_i))}{(x_i - x_{i+1})^2} dx
\end{aligned}
$$

And finally f:

$$
\begin{aligned}
f(\varphi^i) &= \int_{-\pi}^{\pi} e^{\sin(x)+\cos(x)} \varphi^i(x)dx \\
&= \int_{x_{i-1}}^{x_i} e^{\sin(x)+\cos(x)} \frac{x - x_{i-1}}{x_i - x_{i-1}} dx + \int_{x_i}^{x_{i+1}} e^{\sin(x)+\cos(x)} \frac{x_{i+1} - x}{x_{i+1} - x_i} dx
\end{aligned}
$$

Putting this together, we get the following coefficient matrix, $M_{i,j}$, such that:

$$M_{i,j} = p_2(\varphi^i, \varphi^j) + p_0(\varphi^i, \varphi^j)$$

And for the right hand side, we get b_i such that:

$$b_i = f(\varphi^i)$$

And so the final problem is to solve for some coefficients u^i such that:

$$Mu = b$$

c To evaluate the integrals in part b, we use numerical quadrature (except for the Laplacian term, which is trivial). Here is my code (written in python using SciPy, PyLab and Sympy):

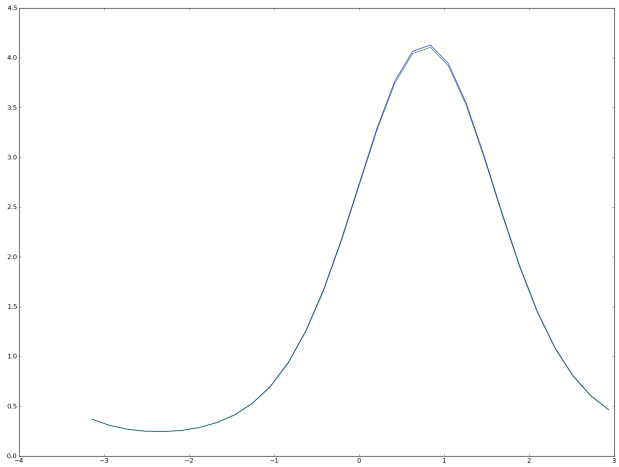
```
from scipy import *
from scipy.sparse import *
from scipy.linsolve import *
from pylab import *
import sympy as sp
import sympy.abc as abc
import sympy.integrals as spi

def fe_solve(x, F2, F0, FR, X):
```

```
    M = len(x)
    A = dok_matrix((M, M))
    b = zeros(M)

    def delta(i,j):
        return int((i + M)%M == (j+M)%M)

    def p2(i, j, xi0, xi1, xi2):
        return F2 * (delta(i, j+1) - delta(i,j)) / (xi1 - xi0) + \
               F2 * (delta(i, j-1) - delta(i,j)) / (xi2 - xi1)

    def p0(i, j, xi0, xi1, xi2):
        t1 = spi.Integral(F0 * (X - xi0) * ( \
              delta(i,j)   * (X - xi0) \
            + delta(i,j+1) * (xi1 - X)), (X, xi0, xi1)).n()  \
            / (xi1 - xi0)**2
        t2 = spi.Integral(F0 * (xi2 - X) * ( \
              delta(i,j)   * (xi2 - X) \
            + delta(i,j-1) * (X - xi1)), (X, xi1, xi2)).n()  \
            / (xi2 - xi1)**2
        return t1 + t2

    def f(i, xi0, xi1, xi2):
        return spi.Integral(FR * (X - xi0), (X, xi0, xi1)).n()  / (xi1 - xi0) \
             + spi.Integral(FR * (xi2 - X), (X, xi1, xi2)).n()  / (xi2 - xi1)

    for i in range(M):
        #Handle periodicity conditions
        xi0 = x[i-1]
        xi1 = x[i]
        xi2 = x[(i+1) % len(x)]
        if(xi0 > xi1):
            xi0 -= 2 * pi
        if(xi1 > xi2):
            xi2 += 2 * pi

        #Construct A
        for j in range(M):
            t = p2(i, j, xi0, xi1, xi2) + p0(i, j, xi0, xi1, xi2)
            if(abs(t) > 1e-6):
                A[i,j] = float(t)

        #Construct b
        b[i] = f(i, xi0, xi1, xi2)

    return spsolve(A, b)

#Use sympy to create symbolic coefficients
X = abc.x
F2 = 1      # Must be a scalar
F0 = sp.sin(X) + sp.cos(X) + sp.sin(2 * X)
FR = sp.exp(sp.sin(X) + sp.cos(X))

#Generate grid and solve U
x = arange(-pi, pi, 2. * pi / 30.)
U = fe_solve(x, F2, F0, FR, X)

#Compute exact soln. for comparison
Uex = exp(sin(x) + cos(x))

#Plot
P = figure()
plot(x, U, figure=P)
plot(x, Uex, figure=P)
P.savefig("prob1_plot.png")

#Calculate the L2 error norm
def get_error(M):
    x = arange(-pi, pi, 2. * pi / M)
    U = fe_solve(x, F2, F0, FR, X)
    Uex = exp(sin(x) + cos(x))
    return norm(U - Uex)

#Plot relative errors
N = range(5, 120, 5)
E = [get_error(M) for M in N]
Q = figure()
```

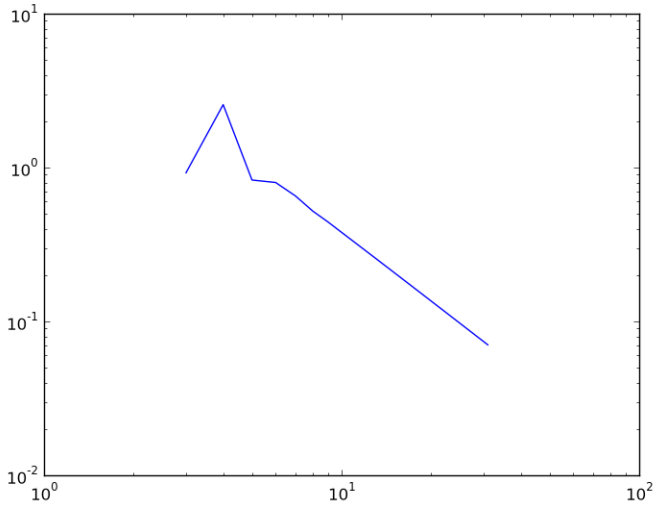
```
loglog(N, E, figure=Q)
Q.savefig("prob1_err.png")
```

prob1.py

And here is the output plot of the computed solution compared to the exact solution, $u(x) = \exp(\sin(x) + \cos(x))$:



And here is the output log-log plot of the L2 error:



2

a To start with, we modify p_0 and f from part 1, giving the following new form for M and b:

$$M_{i,j} =$$

And:

$$b_i =$$

To compute the error within the i^{th} node we do 'blah'

b

c