S. Mishra
L. Scarabosio
J. Sukys

Spring Term 2014

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 2

Introduction. The first problem of this problem sheet concerns the Finite Difference discretization of the Poisson equation on a 2-dimensional domain.
The other two problems address the discretization of one-dimensional problems using the Finite-Element method.

Problem 2.1 Finite Differences for 2D Poisson Equation

In this problem we consider the Finite Differences discretization of the Poisson problem on the unit square:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega := (0,1)^2, \\ u &= 0 \quad \text{on } \partial\Omega, \end{aligned} \tag{2.1.1}$$

for a bounded and continuous function $f \in \mathcal{C}^0(\overline{\Omega})$.

We consider a regular tensor product grid with meshwidth $h := (N+1)^{-1}$ and we assume a lexikographic numbering of the interior vertices of the mesh as depicted in Fig.2.1.

We first consider the 5-point stencil finite difference scheme for the operator $-\Delta$ described by the 5-points stencil shown in Fig. 2.2.

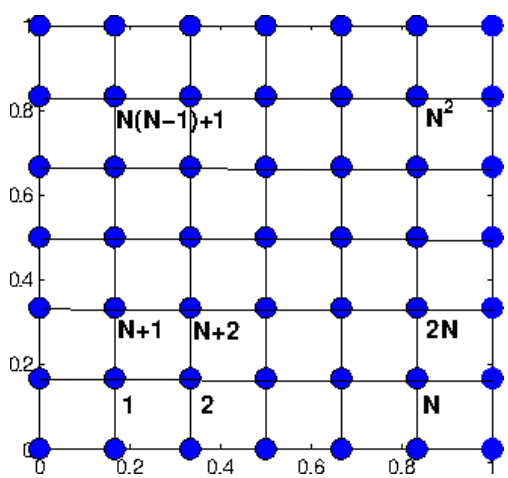

Figure 2.1: Lexikographic numbering of vertices of the equidistant tensor product mesh.

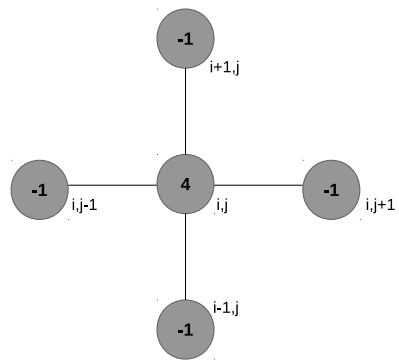

Figure 2.2: 5-point stencil for subproblems (2.1a)-(2.1f).

(2.1a) Write the system

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{L} \tag{2.1.2}$$

corresponding to the discretization of (2.1.1) using the stencil in Fig. 2.2, specifying the matrix $\boldsymbol{A}$ and the vectors $\boldsymbol{L}$ and $\boldsymbol{u}$.

Solution: We have a block tridiagonal matrix $\boldsymbol{A} \in \mathbb{R}^{N^2 \times N^2}$

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{B} & -\boldsymbol{I} & 0 & \dots & \dots & 0 \\ -\boldsymbol{I} & \boldsymbol{B} & -\boldsymbol{I} & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & -\boldsymbol{I} & \boldsymbol{B} & -\boldsymbol{I} \\ 0 & \dots & \dots & 0 & -\boldsymbol{I} & \boldsymbol{B} \end{pmatrix}$$

where $\boldsymbol{I} \in \mathbb{R}^{N \times N}$ is the identity matrix, and $\boldsymbol{B} \in \mathbb{R}^{N \times N}$ is the tridiagonal matrix

$$\boldsymbol{B} = \begin{pmatrix} 4 & -1 & 0 & \dots & \dots & 0 \\ -1 & 4 & -1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & 0 & -1 & 4 \end{pmatrix}$$

The vectors $\boldsymbol{L}$ and $\boldsymbol{u}$ are given by $(\boldsymbol{L})_j = f(x_j)$ and $(\boldsymbol{u})_j = u_h(x_j)$, where j is the node index according to the lexikographic order of Fig. 2.1 and u_h denotes the discrete solution.

(2.1b) Write a function

```
A = PoissonMatrix2D(N)
```

to construct the matrix $\boldsymbol{A}$ in (2.1.2), where N denotes the number of interior grid points along one dimension.

Solution: See Listing 2.1.

Listing 2.1: Implementation for `PoissonMatrix2D`

```
function A = PoissonMatrix2D(N)

e1 = ones(N^2,1);
e2 = repmat([ones(N-1,1);0],N,1);
e3 = repmat([0;ones(N-1,1)],N,1);
A = spdiags([-e1 -e2 4*e1 -e3 -e1],[-N -1 0 1 N],N^2,N^2);

end
```

(2.1c) Write a function

`L = PoissonRHS2D(FHandle,N)`

to build the vector $\boldsymbol{L}$ in (2.1.2). Here `FHandle` is a function handle to the function f in (2.1.1) and N the number of interior grid points.

Solution: See Listing 2.2.

Listing 2.2: Implementation for `PoissonRHS2D`

```
function L = PoissonRHS2D(FHandle,N)

x = linspace(0,1,N+2);
x = x(2:end-1);
[X,Y] = meshgrid(x);
Coords = [Y(:) X(:)];

L = 1/((N+1)^2)*FHandle(Coords);

end
```

(2.1d) Write a function

`uh = PoissonSolve2D(FHandle,N)`

to solve the system (2.1.2), with `FHandle` and N as in the previous subproblems.

Solution: See Listing 2.3.

Listing 2.3: Implementation for `PoissonSolve2D`

```
function u = PoissonSolve2D(FHandle,N)

u = zeros((N+2)^2,1);
Dofs = (1:(N+2)^2)';
FreeDofs = [];
for i=2:N+1
    FreeDofs = [FreeDofs;
       Dofs((N+2)*(i-1)+2:(N+2)*(i-1)+N+1)];
```

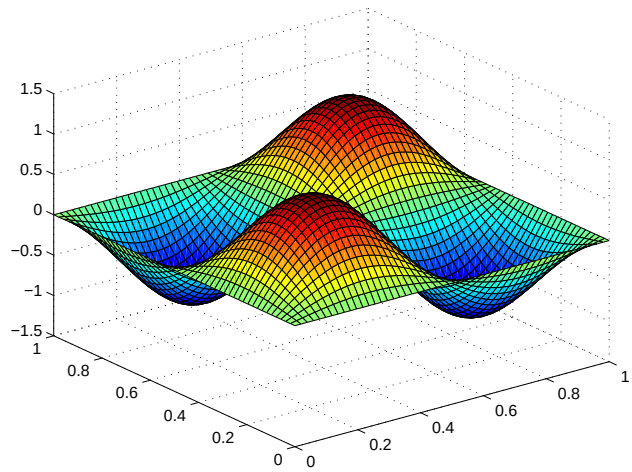

Figure 2.3: Plot for subproblem (2.1e).

```
end
A = PoissonMatrix2D(N);
L = PoissonRHS2D(FHandle,N);
u(FreeDofs) = A\L;

end
```

(2.1e) Plot the discrete solution that you get from subproblem (2.1d) for $f(x, y) = 8\pi^2 \sin(2\pi x) \sin(2\pi y)$ and $N = 50$.

Solution: See Fig. 2.3.

(2.1f) Investigate the convergence of the scheme in the L^∞-norm for $N = 10 \cdot 2^r$, $r = 1, \ldots, 6$. As right-handside f, consider the function $f(x, y) = 8\pi^2 \sin(2\pi x) \sin(2\pi y)$.
Which rate do you observe?

HINT: The exact solution is $u(x, y) = \sin(2\pi x) \sin(2\pi y)$.

Solution: See Listing 2.4 for the code, and Figure 2.4 for the plots. The convergence is algebraic, with rate about 2.

Listing 2.4: Error computation for (2.1f)

```
N = 20;

UHandle = @(x) sin(2*pi*x(:,1)).*sin(2*pi*x(:,2));
FHandle = @(x) 8.*pi^2.*UHandle(x);

h=[];
err=[];

for i=1:6
   u = PoissonSolve2D(FHandle,N);
```

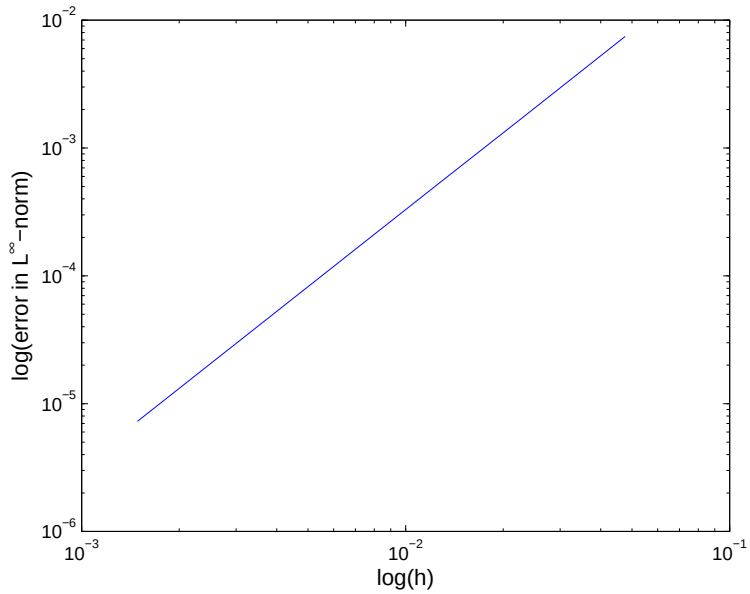

Figure 2.4: Error plot for subproblem (2.1f).

```
    x = linspace(0,1,N+2);
    [X,Y] = meshgrid(x);
    uex = UHandle([Y(:) X(:)]);
    h = [h 1/(N+1)];
    err = [err max(abs(u-uex))];
    N = (N+1)*2-1;
end
```

At this point, we are unsatisfied about the performance of the scheme that we consider, and we want then to adopt an higher order Finite Difference scheme.
To this aim, we consider the so-called *Collatz Merhstellenverfahren*, described by the 9-point stencil depicted in Fig. 2.5.

(2.1g) Describe the matrix $\boldsymbol{A}_M$ resulting from the discretization of (2.1.1) using the stencil 2.5.

Solution: The matrix $\boldsymbol{A}_M$ associated to the Mehrstellenverfahren is a banded matrix given by

$$\boldsymbol{A}_M = \begin{pmatrix} \boldsymbol{B}_1 & -\boldsymbol{I} & 0 & \dots & \dots & \dots & 0 \\ \boldsymbol{B}_3 & \boldsymbol{B}_2 & \boldsymbol{B}_3 & \boldsymbol{B}_4 & 0 & \dots & 0 \\ \boldsymbol{B}_4 & \boldsymbol{B}_3 & \boldsymbol{B}_2 & \boldsymbol{B}_3 & \boldsymbol{B}_4 & \cdots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & \dots & \boldsymbol{B}_4 & \boldsymbol{B}_3 & \boldsymbol{B}_2 & \boldsymbol{B}_3 & \boldsymbol{B}_4 \\ 0 & \dots & 0 & \boldsymbol{B}_4 & \boldsymbol{B}_3 & \boldsymbol{B}_2 & \boldsymbol{B}_3 \\ 0 & \dots & \dots & \dots & 0 & -\boldsymbol{I} & \boldsymbol{B}_1 \end{pmatrix},$$

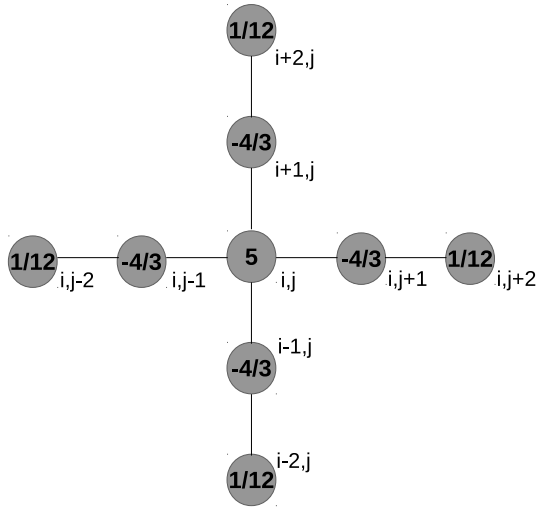

Figure 2.5: 5-point stencil for subproblems (2.1g)-(2.1j).

with $\boldsymbol{I} \in \mathbb{R}^{N\times N}$ the identity matrix and

$$\boldsymbol{B}_1 = \begin{pmatrix} 4 & -1 & 0 & \dots & \dots & 0 \\ -1 & 4 & -1 & 0 & \dots & 0 \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ 0 & \dots & 0 & -1 & 4 & -1 \\ 0 & \dots & \dots & 0 & -1 & 4 \end{pmatrix} \in \mathbb{R}^{N\times N} \quad \boldsymbol{B}_2 = \begin{pmatrix} 4 & -1 & 0 & \dots & \dots & \dots & 0 \\ -\frac{4}{3} & 5 & -\frac{4}{3} & \frac{1}{12} & 0 & \dots & 0 \\ \frac{1}{12} & -\frac{4}{3} & 5 & -\frac{4}{3} & \frac{1}{12} & \dots & 0 \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ 0 & \dots & \frac{1}{12} & -\frac{4}{3} & 5 & -\frac{4}{3} & \frac{1}{12} \\ 0 & \dots & 0 & \frac{1}{12} & -\frac{4}{3} & 5 & -\frac{4}{3} \\ 0 & \dots & \dots & \dots & 0 & -1 & 4 \end{pmatrix} \in \mathbb{R}^{N\times N}$$

$$\boldsymbol{B}_3 = \begin{pmatrix} -1 & & & & \\ & -\frac{4}{3} & & & \\ & & \ddots & & \\ & & & -\frac{4}{3} & \\ & & & & -1 \end{pmatrix} \in \mathbb{R}^{N\times N} \quad \boldsymbol{B}_4 = \begin{pmatrix} 0 & & & & \\ & \frac{1}{12} & & & \\ & & \ddots & & \\ & & & \frac{1}{12} & \\ & & & & 0 \end{pmatrix} \in \mathbb{R}^{N\times N}$$

(2.1h) Write a function

```
A = PoissonMehrstellen(N)
```

to implement the matrix $\boldsymbol{A}_M$ from subproblem (2.1g). Here N denotes again the number of interior grid points along one dimension.

Solution: See Listing 2.5.

Listing 2.5: Implementation for `PoissonMehrstellen`

```
function A = PoissonMehrstellen(N)

```

```
e = ones(N,1);
e2 = [4;5*ones(N-2,1);4];
d1 = [4*e;repmat(e2,N-2,1);4*e];

e3 = [0;ones(N-1,1)];
e4 = [0;-1;-4/3*ones(N-2,1)];
d2 = [-e3;repmat(e4,N-2,1);-e3];

e5 = [ones(N-1,1);0];
e6 = [-4/3*ones(N-2,1);-1;0];
d3 = [-e5;repmat(e6,N-2,1);-e5];

e7 = [0;ones(N-3,1);0;0];
e8 = [0;0;ones(N-3,1);0];
d4 = [zeros(N+2,1);repmat(1/12*e7,N-2,1);zeros(N-2,1)];
d5 = [zeros(N-2,1);repmat(1/12*e8,N-2,1);zeros(N+2,1)];

e9 = [-1; -4/3*ones(N-2,1);-1];
d6 = [-e;-e;repmat(e9,N-2,1);];
d7 = [repmat(e9,N-2,1);-e;-e];

e10 = [0;1/12*ones(N-2,1);0];
d8 = [zeros(3*N,1);repmat(e10,N-3,1)];
d9 = [repmat(e10,N-3,1);zeros(3*N,1)];

A = spdiags([d9 d7 d5 d3 d1 d2 d4 d6 d8],[-2*N -N -2 -1 0 1 2
   N 2*N],N^2,N^2);

end
```

(2.1i) Write a function

```
uh = PoissonSolveMehrstellen(FHandle,N)
```

to solve the system (2.1.1) using the *Mehrstellenverfahren*. The arguments are the same as in subproblem (2.1d).

Solution: See Listing 2.6.

Listing 2.6: Implementation for `PoissonSolveMehrstellen`

```
function u = PoissonSolveMehrstellen(FHandle,N)

u = zeros((N+2)^2,1);
Dofs = (1:(N+2)^2)';
FreeDofs = [];
for i=2:N+1
    FreeDofs = [FreeDofs;
       Dofs((N+2)*(i-1)+2:(N+2)*(i-1)+N+1)];
```

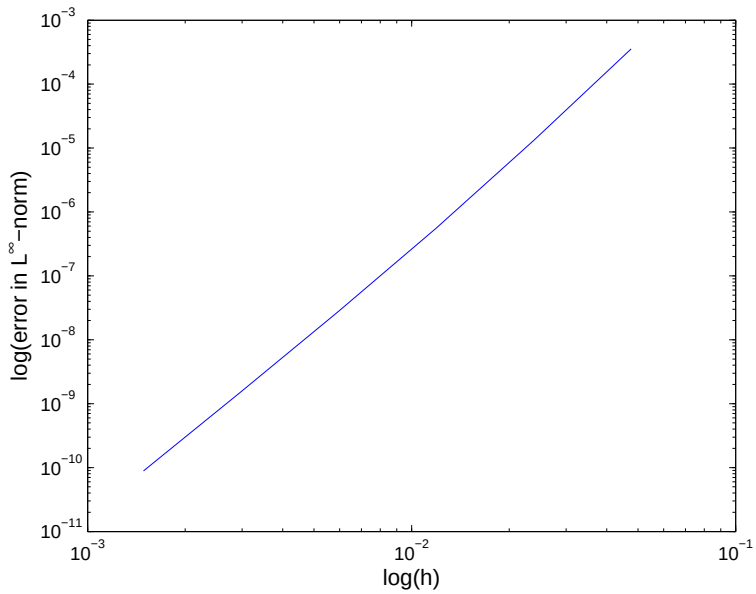

Figure 2.6: Error plot for subproblem (2.1j).

```
end
A = PoissonMehrstellen(N);
L = PoissonRHS2D(FHandle,N);
u(FreeDofs) = A\L;

end
```

(2.1j) Investigate the convergence of the *Mehrstellenverfahren* in the L^∞-norm for $N = 10 \cdot 2^r$, $r = 1, \ldots, 6$. Consider the same right-handside as in subproblem (2.1f).
Which rate of convergence do you observe?

Solution: See Listing 2.7 for the code, and Figure 2.6 for the plots. The convergence is algebraic, with rate above 4 (≈ 4.3757). Indeed, it can be proven theoretically that the Mehrstellenverfahren is a fourth-order method.

Listing 2.7: Error computation for (2.1f)

```
N = 20;

UHandle = @(x) sin(2*pi*x(:,1)).*sin(2*pi*x(:,2));
FHandle = @(x) 2.*(2*pi)^2.*UHandle(x);

h=[];
err=[];

for i=1:6
   u = PoissonSolveMehrstellen(FHandle,N);
   x = linspace(0,1,N+2);
   [X,Y] = meshgrid(x);
```

```
    uex = UHandle([Y(:) X(:)]);
    h = [h 1/(N+1)];
    err = [err max(abs(u-uex))];
    N = (N+1)*2-1;
end
```

Listing 2.8: Testcalls for Problem 2.1

```
UHandle = @(x) sin(2*pi*x(:,1)).*sin(2*pi*x(:,2));
FHandle = @(x) 8.*pi^2.*UHandle(x);

fprintf('\n\n##PoissonSolve2D:')
PoissonSolve2D(FHandle,5)'

fprintf('\n\n##PoissonSolveMehrstellen:')
PoissonSolveMehrstellen(FHandle,5)'
```

Listing 2.9: Output for Testcalls for Problem 2.1

```
>>test_call

##PoissonSolve2D:
ans =

  Columns 1 through 15

    0.8225    0.8225    0.0000   -0.8225   -0.8225    0.8225
       0.8225    0.0000   -0.8225   -0.8225    0.0000    0.0000
       -0.0000   -0.0000   -0.0000

  Columns 16 through 25

   -0.8225   -0.8225   -0.0000    0.8225    0.8225   -0.8225
      -0.8225   -0.0000    0.8225    0.8225

##PoissonSolveMehrstellen:
ans =

  Columns 1 through 15

    0.8177    0.8128    0.0000   -0.8128   -0.8177    0.8128
       0.7888    0.0000   -0.7888   -0.8128    0.0000    0.0000
       0.0000   -0.0000   -0.0000

  Columns 16 through 25

   -0.8128   -0.7888   -0.0000    0.7888    0.8128   -0.8177
      -0.8128   -0.0000    0.8128    0.8177
```

Problem 2.2 Linear Finite Elements in 1D (Core problem)

In class the Galerkin discretization of a 2-point boundary value problem by means of trial and test spaces of merely continuous piecewise linear functions was discussed. The Galerkin matrix for a linear variational problem was derived in detail. In this problem we practise the crucial steps for the slightly modified linear variational problem

$$u \in H_0^1([a,b]): \quad \int_a^b \frac{\mathrm{d}u}{\mathrm{d}x}(x)\,\frac{\mathrm{d}v}{\mathrm{d}x}(x) + c\,u(x)v(x)\,\mathrm{d}x = \int_a^b g(x)v(x)\;\mathrm{d}x, \quad \forall v \in H_0^1([a,b]), \tag{2.2.1}$$

where $c > 0$, $-\infty < a < b < \infty$, $g \in \mathcal{C}^0([a,b])$. Please note that both trial and test functions vanish at the endpoints of the interval, as indicated by the subscript "0" in the symbol for the function space.

(2.2a) Derive the Galerkin matrix for (2.2.1), when using the trial and test space $\mathcal{S}_{1,0}^0(\mathcal{M})$ of continuous, piecewise linear functions on an *equidistant* mesh $\mathcal{M}$ with $N \in \mathbb{N}$ interior nodes. The standard basis of tent functions is to be used.

Solution: Let $\Omega = [a,b]$ and $\mathcal{M} := \{(x_{j-1}, x_j) \,|\, 0 \le j \le N\}$ be an equidistant mesh, meaning that $x_j = a + jh$ with $h = \frac{b-a}{N}$.

We first write the discretized version of the equation:

$$\int_a^b \frac{\mathrm{d}u_N}{\mathrm{d}x}(x)\frac{\mathrm{d}v_N}{\mathrm{d}x}(x) + cu_N(x)v_N(x) = \int_a^b g(x)v_N(x)\,\mathrm{d}x$$

which, written in the basis $\{b_N^i, 1 \le i \le N\}$ of tent function, becomes

$$\sum_{i=0}^{N}\left(\int_a^b \frac{\mathrm{d}b_N^i}{\mathrm{d}x}(x)\frac{\mathrm{d}b_N^k}{\mathrm{d}x}(x)\,\mathrm{d}x + c\int_a^b b_N^i(x)b_N^k(x)\,\mathrm{d}x\right)\mu_i = \int_a^b g(x)b_N^k(x)\,\mathrm{d}x,\; k \in \{1, ..., N\}$$

This can be translated to a system of linear equations of the form $(\mathbf{A} + c\mathbf{B})\boldsymbol{\mu} = \boldsymbol{\varphi}$ where

$$\mathbf{A}_{i,k} = \int_a^b \frac{\mathrm{d}b_N^i}{\mathrm{d}x}(x)\frac{\mathrm{d}b_N^k}{\mathrm{d}x}(x)\,\mathrm{d}x,$$

$$\mathbf{B}_{i,k} = c\int_a^b b_N^i(x)b_N^k(x)\,\mathrm{d}x,$$

and

$$\boldsymbol{\varphi}_k = \int_a^b g(x)b_N^k(x)\,\mathrm{d}x.$$

As shown in the lecture notes, $\mathbf{A}$ has the following form:

$$\mathbf{A} = \frac{1}{h}\begin{pmatrix} 2 & -1 & 0 & ... & 0 \\ -1 & 2 & -1 & & \\ 0 & -1 & ... & ... & \\ ... & & ... & 2 & -1 \\ 0 & & & -1 & 2 \end{pmatrix}.$$

In an analogous way, the matrix $\mathbf{B}$ takes the following form:

$$\mathbf{B} = h \begin{pmatrix} 2/3 & 1/6 & 0 & \dots & 0 \\ 1/6 & 2/3 & 1/6 & & \\ 0 & 1/6 & \dots & \dots & \\ \dots & & \dots & 2/3 & 1/6 \\ 0 & & & 1/6 & 2/3 \end{pmatrix},$$

where the coefficients for matrix $\mathbf{B}$ have been computed in the following way:

$$\mathbf{B}_{i,i} = \frac{1}{h^2}\int_{x_{i-1}}^{x_i} (x - x_{i-1})^2 \,\mathrm{d}x + \frac{1}{h^2}\int_{x_i}^{x_{i+1}} (x_{i+1} - x)^2 \,\mathrm{d}x = \frac{2}{3}h$$

$$\mathbf{B}_{i,i-1} = \frac{1}{h^2}\int_{x_{i-1}}^{x_i} (x_i - x)(x - x_{i-1}) \,\mathrm{d}x = \frac{1}{6}h$$

$$\mathbf{B}_{i-1,i} = \frac{1}{6}h$$

The Galerkin matrix is therefore the following:

$$\mathbf{G} = \begin{pmatrix} \frac{2}{h} + c\frac{2}{3}h & -\frac{1}{h} + c\frac{1}{6}h & 0 & \dots & 0 \\ -\frac{1}{h} + c\frac{1}{6}h & \frac{2}{h} + c\frac{2}{3}h & -\frac{1}{h} + c\frac{1}{6}h & & \\ 0 & -\frac{1}{h} + c\frac{1}{6}h & \dots & \dots & \\ \dots & & \dots & \frac{2}{h} + c\frac{2}{3}h & -\frac{1}{h} + c\frac{1}{6}h \\ 0 & & & -\frac{1}{h} + c\frac{1}{6}h & \frac{2}{h} + c\frac{2}{3}h \end{pmatrix}$$

(2.2b) To obtain the right-hand side vector of the linear system arising from the Galerkin discretization of (2.2.1) as described in subproblem (2.2a), one relies on the composite trapezoidal rule on $\mathcal{M} = \{x_0 = a, x_1, \dots, x_N, x_{N+1} = b\}$ for numerical quadrature:

$$\int_a^b f(x)\,\mathrm{d}x \approx f(a)\frac{h}{2} + h\sum_{i=1}^{N} f(x_i) + f(b)\frac{h}{2}, \tag{2.2.2}$$

with h the meshsize.

Implement an efficient function

```
u = linfegalerkinsol(a,b,c,g,N)
```

that computes the values of the Galerkin solution $u_N \in \mathcal{S}^0_{1,0}(\mathcal{M})$ at the nodes of the mesh $\mathcal{M}$ and returns them in the row vector `u`. The arguments `a`, `b`, `c` supply the domain $\Omega = [a, b]$, and the coefficient $c > 0$, whereas `g` is a function handle to the source function g. The argument `N` passes the number of interior nodes of the equidistant mesh.

Solution: Using the trapezoidal rule, one gets

$$\boldsymbol{\varphi}_k = \int_a^b g(x) b_N^k(x)\,\mathrm{d}x \simeq h\hat{\sum}_{i=0}^{N} g(x_i) b_N^k(x_i),$$

where the symbol $\hat{\sum}$ denotes half weight on the first and last summand. Since $b^k(x_i) \neq 0$ only for $i = k$, the sum collapses nicely into $\boldsymbol{\varphi}_k \simeq hg(x_k)$ for $1 < k < N - 1$ and $\boldsymbol{\varphi}_k \simeq hg(x_k)/2$ for $k = 0$ and $k = N$.

See Listing 2.10 for the code.

Listing 2.10: Linear finite elements in 1D (homogeneous Dirichlet)

```
function u = linfegalerkinsol(a, b, c, g, N)
% Computes the values of the discrete Galerkin solution using as
% basis tent functions on the domain [a,b] for the problem
% Integral(u'v' + cuv) = Integral(gv), for the given function
% handle g and using a mesh with N internal nodes.
% The function is constrained by the following boundary conditions
% u(a)=0 and u(b)=0

% mesh
h = (b-a)/(N+1);
x = linspace(a,b,N+2)';

% compute finite element matrix A and right-hand side phi
v1 = (-1/h + 1/6*c*h)*ones(N+1,1);
v2 = [1/h + 1/3*c*h; (2/h + 2/3*c*h)*ones(N,1); 1/h + 1/3*c*h];
A = gallery('tridiag',v1,v2,v1);
phi = h*[0.5*g(x(1)); g(x(2:end-1)); 0.5*g(x(end))];

% incorporate boundary data
u = zeros(N+2,1);

% solve the linear system
u(2:end-1) = A(2:end-1,2:end-1)\phi(2:end-1);

return
```

(2.2c) State and justify the asymptotic computational complexity of `linfegalerkinsol` in terms of the problem size parameter N.

Solution: The system matrix is *tridiagonal* and s.p.d. and, thus, Gaussian elimination without pivoting/Cholesky factorization can solve it with an asymptotic computational effort of $\mathcal{O}(N)$.

(2.2d) Plot the Galerkin solution u_N for $\Omega := [-\pi, \pi]$, $c = 1$, $g(x) = \sin(x)$, and $N = 50, 100, 200$. To validate your code compare u_N with the exact analytic solution $u(x) = \frac{1}{2}\sin(x)$.

Solution: The resulting graph should look like the following:

(2.2e) Extend your above implementation of `linfegalerkinsol` to

```
u = linfegalerkinsolDirichlet(a,b,c,g,N,ua,ub),
```

where the optional arguments `ua`, `ub` may be used to specify *boundary values* for the solution u of (2.2.1). This means that now we seek to solve (2.2.1) under the constraints $u(a) = u_a$, $u(b) = u_b$.

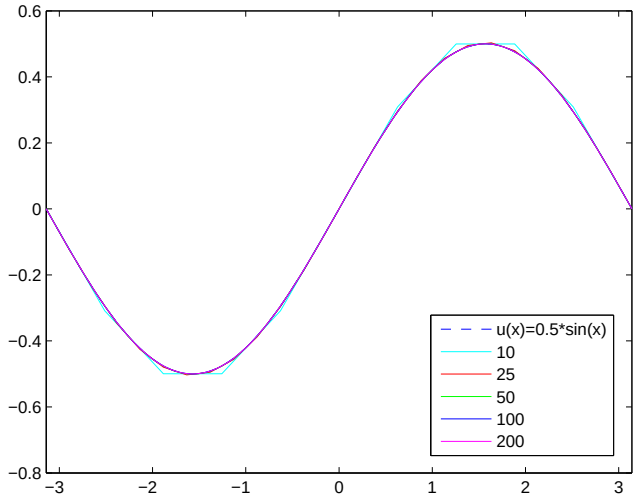

Figure 2.7: Plot for subproblem (2.2d).

HINT: Use the offset function technique to arrive at a modified right-hand side of the linear system of equations that incorporates the values u_a and u_b.

Solution: See Listing 2.11 for the code.

Listing 2.11: Linear finite elements in 1D (inhomogeneous Dirichlet)

```
function u = linfegalerkinsolDirichlet(a, b, c, g, N, ua, ub )
% Computes the values of the discrete Galerkin solution using as
% basis tent functions on the domain [a,b] for the problem
% Integral(u'v' + cuv) = Integral(gv), for the given function
% handle g and using a mesh with N internal nodes.
% The function is constrained by the following boundary conditions
% u(a)=ua and u(b)=ub

% mesh
h = (b-a)/(N+1);
x = linspace(a,b,N+2)';

% compute finite element matrix A and right-hand side phi
v1 = (-1/h + 1/6*c*h)*ones(N+1,1);
v2 = [1/h + 1/3*c*h; (2/h + 2/3*c*h)*ones(N,1); 1/h + 1/3*c*h];
A = gallery('tridiag',v1,v2,v1);
phi = h*[0.5*g(x(1)); g(x(2:end-1)); 0.5*g(x(end))];

% incorporate boundary data
mu = zeros(N+2,1); mu(1) = ua; mu(end) = ub;
phi = phi - A*mu;
```

```
% solve the linear system
mu(2:end-1) = A(2:end-1,2:end-1)\phi(2:end-1);
u = mu;

return
```

(2.2f) Plot the Galerkin solution u_N for $\Omega := [-\pi, \pi]$, $c = 1$, $g(x) = \cos(x)$, $u_a = u_b = -\frac{1}{2}$ and $N = 50$. To validate your code compare u_N with the exact analytic solution $u(x) = \frac{1}{2}\cos(x)$.

Solution: The resulting graph should look like the following:

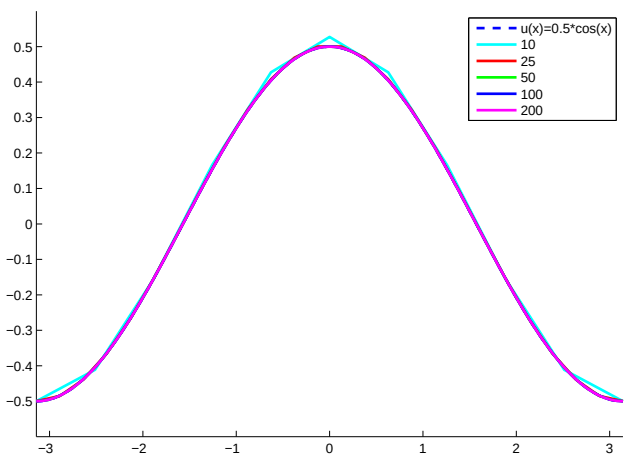

Figure 2.8: Plot for subproblem (2.2f).

(2.2g) Investigate the convergence of the finite element discretization developed in (2.2b) in the L^∞-norm using $N = 10 \cdot 2^r$ degrees of freedom, for $r = 1, 2, \ldots, 9$. This sequence of N-values should also be used for the following sub-problems. Use the test case discussed in (2.2d).

The exact evaluation of the L^∞-norm is hardly ever possible. Thus we have to resign ourselves to evaluating it only approximately. To do so, we rely on a mesh $\widetilde{\mathcal{M}}$ obtained by splitting each cell of the finite element mesh $\mathcal{M}$ into four smaller cells of equal size. Then sample the modulus of the error on all vertices of the finer mesh $\widetilde{\mathcal{M}}$ and find the maximal value.

HINT: For $N = 10$, the error should be around 0.016.

(2.2h) Investigate the convergence in the L^2-norm

$$\|e\|_{L^2} := \left(\int_0^1 |e(x)|^2 \, \mathrm{d}x \right)^{\frac{1}{2}}, \quad e \in \mathcal{C}^0_{\mathrm{pw}}([0,1]). \tag{2.2.3}$$

To evaluate this norm approximately, use *composite Gaussian quadrature* on the mesh $\mathcal{M}$ with two points per mesh cell.

HINT: The nodes and weights for 2-point Gaussian quadrature on $[-1, 1]$ are $\zeta_1 = -\frac{1}{3}\sqrt{3}$, $\zeta_2 = \frac{1}{3}\sqrt{3}$, $\omega_1 = 1$, $\omega_2 = 1$. This quadrature rule has to be transformed to all mesh cells (x_{j-1}, x_j). For $N = 10$, the error should be around 0.012.

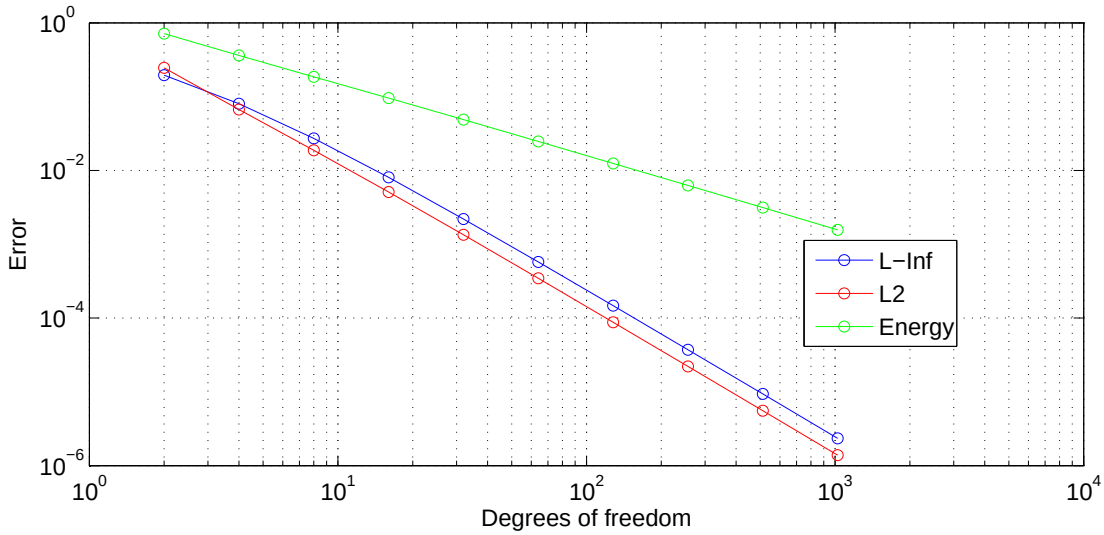

Figure 2.9: Error plots for problem 1.

(2.2i) Investigate the convergence in the energy seminorm

$$|e|_{H^1} := \left(\int_0^1 \left| \frac{\mathrm{d}e}{\mathrm{d}x}(x) \right|^2 \right)^{\frac{1}{2}}, \quad u \in \mathcal{C}^1_{\mathrm{pw}}([0,1]). \tag{2.2.4}$$

Again, use composite 2-point Gaussian quadrature on $\mathcal{M}$ to approximate the integral.

HINT: For $N = 10$, the error should be around 0.150.

Solution: See Listing 2.12 for the code, and Figure 2.9 for the plots. The convergence is algebraic, with rate about 2 for the L^∞- and L^2-errors, and rate 1 for the energy error.

Listing 2.12: Error computation for (2.2g) - (2.2i)

```
function [nvals, linferrs, ltwoerrs, energyerrs] =
   errors_p1(nvals)

    linferrs = zeros(size(nvals));
    ltwoerrs = zeros(size(nvals));
    energyerrs = zeros(size(nvals));

    i = 1;
    for dofs = nvals,

        h = 2*pi/(dofs+1);

        % Compute solution
        extmesh = linspace(-pi, pi, dofs+2);
        extU = linfegalerkinsol(-pi, pi, 1, @sin, dofs);

        % L-Inf error
        xpts = 2*pi*(0:4*dofs)/(4*dofs)-pi;
```

```
        err = abs(linterp(extmesh, extU, xpts) - 0.5*sin(xpts));
        linferrs(i) = max(err);

        % L2 error
        % shift the mesh by h/2 so that we evaluate the error in the middle of each cell.
        xpts = repmat(extmesh(1:length(extmesh)-1)+h/2,2,1); xpts = xpts(:);
        gqpts = repmat([-1;1]/sqrt(3),1,dofs+1)*h/2; gqpts = gqpts(:) + xpts;
        ltwoerrs(i) = sqrt(sum((h/2)*(linterp(extmesh, extU, gqpts) - 0.5*sin(gqpts')).^2));

        % Energy error
        derivs = repmat(extU(2:end) - extU(1:end-1),1,2); derivs = derivs';
        derivs = derivs(:)/h - 0.5*cos(gqpts);
        energyerrs(i) = sqrt(sum((h/2)*derivs.^2));

        i = i+1;

    end;

    clf;
    loglog(nvals, linferrs, 'bo-'); hold on;
    p = polyfit(log(nvals), log(linferrs), 1);
    disp(sprintf('L-Inf error algebraic with rate %f', -p(1)));

    plot(nvals, ltwoerrs, 'ro-');
    p = polyfit(log(nvals), log(ltwoerrs), 1);
    disp(sprintf('L2 error algebraic with rate %f', -p(1)));

    plot(nvals, energyerrs, 'go-');
    p = polyfit(log(nvals), log(energyerrs), 1);
    disp(sprintf('Energy error algebraic with rate %f', -p(1)));

    grid on;
    xlabel('Degrees of freedom');
    ylabel('Error');
    legend('L-Inf', 'L2', 'Energy');

end
```

Listing 2.13: Testcalls for Problem 2.2

```
a=-pi;
b=pi;
c=1;
N=10;

fprintf('\n\n##linfegalerkinsol:')
linfegalerkinsol(a,b,c,@(x)(sin(x)),N)

fprintf('\n\n##linfegalerkinsolDirichlet:')
linfegalerkinsolDirichlet(a,b,c,@(x)(sin(x)),N,-1/2,-1/2)
```

Listing 2.14: Output for Testcalls for Problem 2.2

```
>> test_call

##linfegalerkinsol:
ans =

         0
   -0.2816
   -0.4737
   -0.5155
   -0.3936
   -0.1467
    0.1467
    0.3936
    0.5155
    0.4737
    0.2816
         0

##linfegalerkinsolDirichlet:
ans =

   -0.5000
   -0.4264
   -0.2097
    0.0780
    0.3434
    0.5015
    0.5015
    0.3434
    0.0780
   -0.2097
   -0.4264
   -0.5000
```

Problem 2.3 $L^2(0,1)$-Orthogonal Projection onto Linear Finite Element Space

In this problem we deal with a very simple *linear* variational problem that does not even involve derivatives. For discretization we employ linear finite elements. A careful re-examination of this section is recommended. You will be asked to implement the method and perform a numerical study of its convergence.

The variational problem reads: seek $u \in H^1([0,1])$:

$$\int_0^1 u(x)v(x)\,\mathrm{d}x = \int_0^1 f(x)v(x)\,\mathrm{d}x \quad \forall v \in V := H^1([0,1]). \tag{2.3.1}$$

Remark: Variational problems of this kind are encountered when computing the $L^2(0,1)$-orthogonal projection of f onto subspaces.

To begin with, we consider Galerkin discretization with the subspace $V_N = \mathcal{S}_1^0(\mathcal{M})$ of piecewise linear continuous functions on a general (not necessarily uniform) mesh $\mathcal{M}$ of $[0,1]$. In the following, use tent functions as a basis. Note that the values of the solution in the boundary points $x = 0$, $x = 1$ are **not** fixed. In fact, this would not make any sense.

(2.3a) What is the Galerkin matrix for this problem? Write a function

```
A = galmatrix_tent(mesh)
```

which takes the mesh points (including the boundary points) as input in the row vector `mesh`, computes the Galerkin matrix, and returns it in the sparse matrix `A`.

HINT: The cell sizes h_i must be taken into account.

Solution: The Galerkin matrix should have the format

$$\mathbf{A} = \begin{pmatrix} \frac{h_1}{3} & \frac{h_1}{6} & & & & \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & & & \\ & \frac{h_2}{6} & \frac{h_2+h_3}{3} & \frac{h_3}{6} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{h_{N-1}}{6} & \frac{h_{N-1}+h_N}{3} & \frac{h_N}{6} \\ & & & & \frac{h_N}{6} & \frac{h_N}{3} \end{pmatrix},$$

where $h_1, \ldots, h_N$ are the mesh widths.

Listing 2.15: Implementation for `galmatrix_tent`

```
function A = galmatrix_tent(mesh)
h = mesh(2:end) - mesh(1:end-1);
v1 = h./6;
v2 = [h;0]./3 + [0;h]./3;

A = gallery('tridiag',v1,v2,v1);
end
```

See Listing 2.15.

(2.3b) Write a function

```
L = rhs_tent(mesh,f)
```

which takes as input the same `mesh` vector and a function handle for f, computes the right-hand side vector, and returns it in the column vector `L`. Use the composite trapezoidal rule for numerical quadrature.

HINT: Assume that `f` can take vector arguments to compute f at each mesh point quickly.

Solution: See Listing 2.16.

Listing 2.16: Implementation for `rhs_tent`

```
function rhs = rhs_tent(mesh, f)

    fvals = f(mesh);
    hvals = mesh(2:end) - mesh(1:end-1);

    rhs = ([hvals;0].*fvals + [0;hvals].*fvals)/2;

end
```

(2.3c) Write a function

```
U = l2proj_tent(mesh,f)
```

that solves (2.3.1) approximately based on linear finite element Galerkin discretization. The arguments `mesh` and `f` are the same as before. The column vector `U` should contain the value of the solution at each mesh point.

Solution: See Listing 2.17.

Listing 2.17: Implementation for `l2proj_tent`

```
function U = l2proj_tent(mesh, f)

    A = galmatrix_tent(mesh);
    L = rhs_tent(mesh, f);
    U = A\L;

end
```

(2.3d) Investigate the convergence of the method from Problem 2.3 with equidistant mesh points for $f(x) = \sqrt{x}$ in the L^∞- and L^2-norms. Compute these norms by sampling on a finer mesh or using composite 2-point Gauss quadrature on $\mathcal{M}$, as in (2.3a) and (2.3b). Use $N = 10 \cdot 2^r$ degrees of freedom, for $r = 1, 2, \ldots, 9$. Which is the rate of convergence? Give an explanation about the result that you get.

HINT: The exact solution is $u(x) = \sqrt{x}$. For $N = 10$, the L^∞-error should be around 0.222, and the L^2-error should be around 0.059.

HINT: Note that $f'(x)$ is large for x close to zero and even singular at $x = 0$, for a different f you would observe other results.

Solution:

Listing 2.18: Implementation for `l2proj_tent_eq`

```
exact = @(x) sqrt(x);
Nvals = 10*2.^(1:9);
l2errors = zeros(1,9);
lierrors = zeros(1,9);

for i = 1:9
    mesh = linspace(0,1,Nvals(i))';
    h = 1/(Nvals(i)-1);

    U = l2proj_tent(mesh, exact);

    qpts = sort([mesh(1:end-1) + h/2 - h/(2*sqrt(3));
       mesh(1:end-1) + h/2 + h/(2*sqrt(3))]);
    appr = linterp(mesh, U, qpts);
    ex = exact(qpts);

    l2errors(i) = sqrt(sum((appr-ex').^2)*(h/2));
    lierrors(i) = max(abs(appr-ex'));
end

clf;
loglog(Nvals, l2errors, 'ro-');
hold on;
loglog(Nvals, lierrors, 'bo-');
xlabel('Degrees of freedom');
ylabel('Error');

p = polyfit(log(Nvals), log(l2errors), 1);
-p(1)
p = polyfit(log(Nvals), log(lierrors), 1);
-p(1)
```

See Listing 2.18 and Figure 2.10. The convergence is algebraic, with rate about $1/2$ for the L^∞-error and rate about 1 for the L^2-error.

Listing 2.19: Testcalls for Problem 2.3

```
fprintf('\n\n##galmatrix_tent:')
full(galmatrix_tent([0; 0.1; 0.7; 1]))

fprintf('\n\n##rhs_tent:')
rhs_tent([0; 0.1; 0.7; 1], @(x) x)'

```

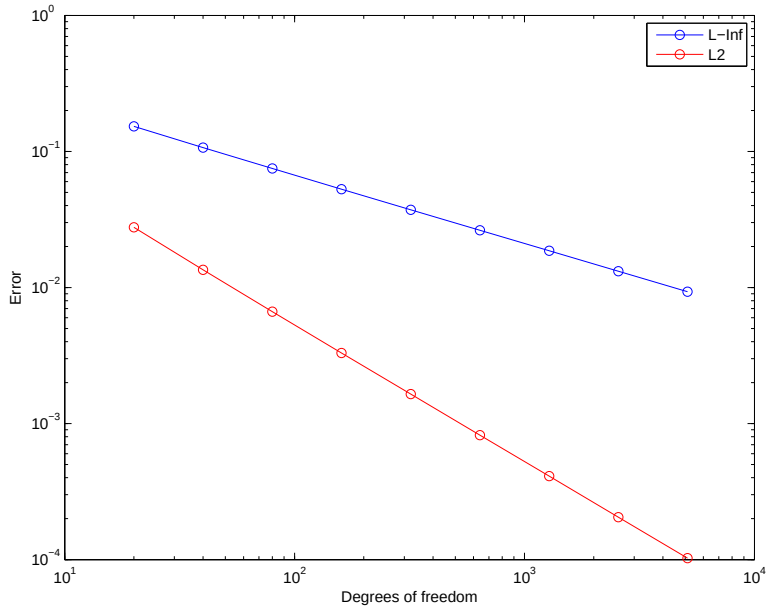

Figure 2.10: Error plots for problem (2.3d).

```
fprintf('\n\n##l2proj_tent:')
l2proj_tent([0; 0.1; 0.7; 1], @(x) x)'
```

Listing 2.20: Output for Testcalls for Problem 2.3

```
>> test_call

##galmatrix_tent:
ans =

    0.0333    0.0167         0         0
    0.0167    0.2333    0.1000         0
         0    0.1000    0.3000    0.0500
         0         0    0.0500    0.1000

##rhs_tent:
ans =

         0    0.0350    0.3150    0.1500

##l2proj_tent:
ans =

    0.1386   -0.2771    0.9735    1.0133
```

Published on March 17.

To be submitted on March 31.

Last modified on March 19, 2014