R. Hiptmair
L. Scarabosio
C. Urzua Torres

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 3

Problem 3.1 Fourier Spectral Galerkin Scheme for Two-Point Boundary Value Problem

In [NPDE, Section 1.5.2.1] you learned about the discretization of 2-point boundary value problems based on global polynomials using integrated Legendre polynomials as basis. The implementation for a linear BVP was presented in [NPDE, § 1.5.48].

This problem is focused on another variant of spectral Galerkin discretization, which relies on non-polynomial trial and test spaces and, again, employs globally supported basis functions. This time the solution will be approximated by linear combinations of trigonometric functions. You will be asked to implement the Galerkin discretization in MATLAB and to study its convergence in a numerical experiment, *cf* [NCSE, Thm. 9.1.4].

We consider the linear variational problem: seek $u \in \mathcal{C}^1_{0,\mathrm{pw}}([0,1])$ such that

$$\int_0^1 \sigma(x) \frac{\mathrm{d}u}{\mathrm{d}x}(x)\, \frac{\mathrm{d}v}{\mathrm{d}x}(x)\ \mathrm{d}x = \int_0^1 f(x)v(x)\ \mathrm{d}x, \quad \forall v \in \mathcal{C}^1_{0,\mathrm{pw}}([0,1]), \tag{3.1.1}$$

cf. [NPDE, Eq. (1.4.23)]. For the discretization of (3.1.1) we may use a so-called *Fourier-spectral* Galerkin method, which boils down to a Galerkin method using the trial and test space

$$V_{N,0} := \operatorname{span}\{\sin(\pi x), \sin(2\pi x), \ldots, \sin(N\pi x)\}\ , \tag{3.1.2}$$

and the basis function already given in the definition (3.1.2). It is related the spectral Galerkin scheme discussed in [NPDE, Section 1.5.2.1].

(3.1a) Show that the functions specified in (3.1.2) really provides a basis of $V_{N,0}$.

HINT: First establish orthogonality of the basis functions with respect to the inner product

$$(f,g)_{L^2(0,1)} := \int_0^1 f(x)g(x)\,\mathrm{d}x\ .$$

Then recall that a basis of a finite dimensional vector space is a maximal *linearly independent* subset.

Solution: We only have to show that the sines are linearly independent. In fact, they are orthogonal. For $i \neq j$:

$$\int_0^1 \sin(i\pi x)\sin(j\pi x)\,\mathrm{d}x = \frac{1}{2}\int_0^1 [\cos((i-j)\pi x) - \cos((i+j)\pi x)]\,\mathrm{d}x = 0,$$

as both $i+j$ and $i-j$ are nonzero.

(3.1b) Which basis should be used for the Fourier spectral scheme, if we had to approximate functions in the space $\mathcal{C}^1_{0,\text{pw}}([a,b])$ for fixed $a < b$, instead of the space $\mathcal{C}^1_{0,\text{pw}}([0,1])$.

HINT: Read [NPDE, § 1.5.41].

Solution: The transformed basis in this case would be

$$V_{N,0}([a,b]) = \text{span}\{\sin(\pi(x-a)/h), \sin(2\pi(x-a)/h), \ldots, \sin(N\pi(x-a)/h)\},$$

with $h = b - a$.

(3.1c) Since (3.1.1) is a linear variational problem, any Galerkin discretization will lead to a linear system of equations. For the case $\sigma \equiv 1$ compute its matrix for the Galerkin scheme relying on $V_{N,0}$ and the trigonometric basis specified in (3.1.2). Discuss structural properties of the matrix like sparsity, symmetry, regularity.

Solution: We need to compute the matrix entries

$$\mathbf{A}_{i,j} = ij\pi^2 \int_0^1 \cos(i\pi x)\cos(j\pi x)\,\mathrm{d}x.$$

Using the identity

$$\cos(i\pi x)\cos(j\pi x) = \frac{1}{2}[\cos((i-j)\pi x) + \cos((i+j)\pi x)],$$

we see as in (3.1a) that the off-diagonal terms are all zero. Then

$$\mathbf{A}_{i,i} = \frac{i^2\pi^2}{2}\int_0^1 (1+\cos(2i\pi x))\,\mathrm{d}x = \frac{i^2\pi^2}{2}.$$

So

$$\mathbf{A} = \frac{\pi^2}{2}\begin{pmatrix} 1 & & & & \\ & 4 & & & \\ & & 9 & & \\ & & & \ddots & \\ & & & & N^2 \end{pmatrix}.$$

This matrix is diagonal, thus both sparse and symmetric.

(3.1d) Write a MATLAB function

```
function A = getGalMat(sigma,N)
```

that computes the Galerkin matrix for the Fourier spectral Galerkin scheme for (3.1.1). Here `sigma` is a handle to the coefficient function σ. The evaluation of the integrals should be done by means of a $3N$-point Gaussian quadrature formula on $[0,1]$.

HINT: The nodes and weights of the Gaussian quadrature rules on $[a,b]$ can be computed by the MATLAB function `[x,w] = gauleg(a,b,n,tol)`, which is available for download.

Solution: See Listing 3.1.

Listing 3.1: Computation of the Galerkin matrix

```
function C = getGalMat(sigma, N)

    [x,w] = gauleg(0,1,3*N);

    C = cos(pi*x*(1:N));
    % C(k,i) = cos(pi i x_k)

    D = bsxfun(@times, C, sigma(x));
    D = bsxfun(@times, D, w);
    % D(k,i) = cos(pi i x_k) sigma(x_k) w_k

    C = C' * D;
    % C(i,j) = sum_k cos(pi i x_k) cos(pi j x_k) sigma(x_k)
       w_k

    C = bsxfun(@times, C, 1:N);
    C = bsxfun(@times, C, (1:N)');
    C = pi^2 * C;

end
```

(3.1e) Write a function

```
function phi = getrhsvector(f,N)
```

that computes the right-hand side vector for the Fourier spectral Galerkin discretization with N basis functions. The routine should rely on $3N$-point Gaussian quadrature for the evaluation of the integrals.

Solution: See Listing 3.2.

Listing 3.2: Computation of the right-hand side vector

```
function phi = getrhsvector(f, N)

    [x,w] = gauleg(0,1,3*N);

    C = sin(pi*x*(1:N));
    C = bsxfun(@times, C, f(x));
    phi = C' * w;

end
```

(3.1f) For $\sigma(x) = \frac{1}{\cosh(\sin(\pi x))}$, $f(x) = \pi^2 \sin(\pi x)$, determine an approximate solution of (3.1.1) by means of the Fourier spectral scheme introduced above. Create a suitable plot of the L^2-norm and L^∞-norm of the discretization error versus the number N of unknowns for $N = 2, 3, \ldots, 14$ and, thus, investigate the convergence of the method [NCSE, Remark 9.1.4].

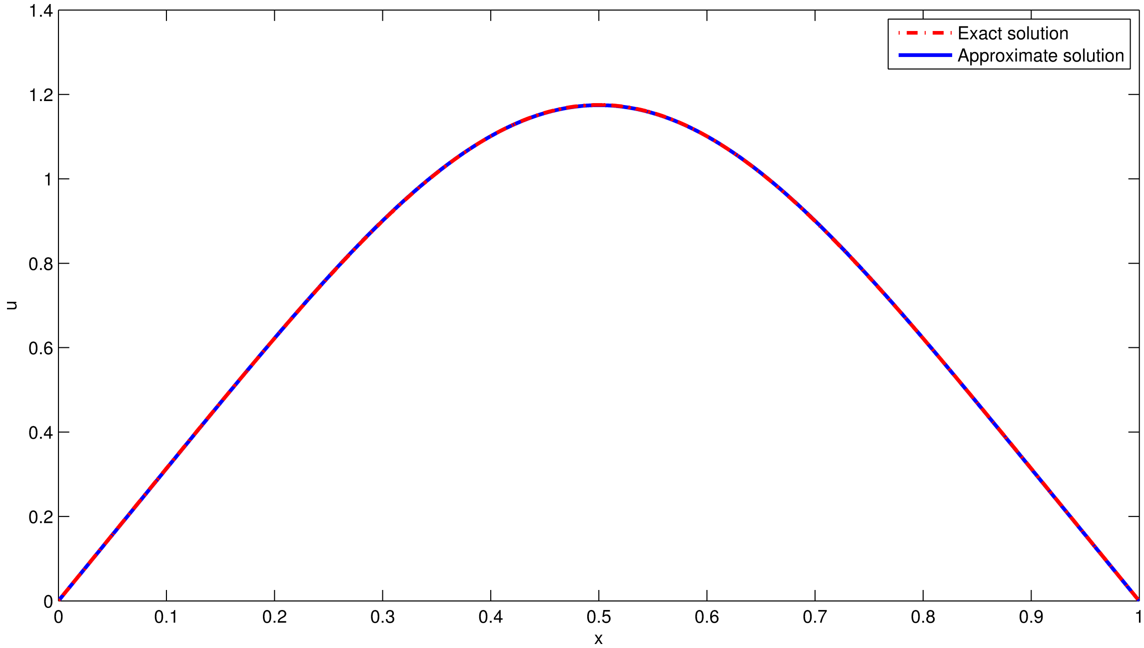


Figure 3.1: Exact and approximate solution for $N = 3$.

The computation of the norms should be done approximately by means of numerical quadrature (equidistant trapeziodal rule with 10^5 points) and sampling (in 10^5 equidistant points), respectively, see [NPDE, Rem. 1.6.19].

HINT: The exact solution of the 2-point boundary value problem is

$$u(x) = \sinh(\sin(\pi x)).$$

Solution:

See Figure 3.1 for the approximate and exact solution at $N = 3$, Figure 3.2 for the convergence plots and Listing 3.3 for the code used.

To compute the $L^2([a, b])$-error (here $[a, b] = [0, 1]$) we use a composite trapeziodal rule with $N + 1$ equidistant points $x_0, \dots, x_N$

$$\int_b^a f(x)\,\mathrm{d}x \approx \frac{b-a}{N}\left(\frac{f(x_0)}{2} + \sum_{i=1}^{N-1} f(x_i) + \frac{f(x_N)}{2}\right) = \frac{b-a}{N}\sum_{i=1}^{N} f(x_i),$$

where in the last step we used the periodicity of f, i.e., $f(x_0) = f(a) = f(b) = f(x_N)$.

To estimate the rate of the exponential convergence, we do the following consideration for the error e

$$e \approx \mathrm{e}^{-\gamma n^\delta} \iff \log e \approx -\gamma n^\delta \iff \log\log e \approx \log(-\gamma) + \log(n^\delta) = \delta \log n + \log -\gamma$$

Then we do a polyfit of the last term and obtain the best matching polynomial in $x = \log n$ and obtain $\delta x + x_0$, and we recompute γ as $\gamma = -\mathrm{e}^{x_0}$. We implement this in Listing 3.3 and obtain a similar convergence for both norms

```
L-inf convergence exponential with gamma:  1.4904 and delta:  1.2291
```

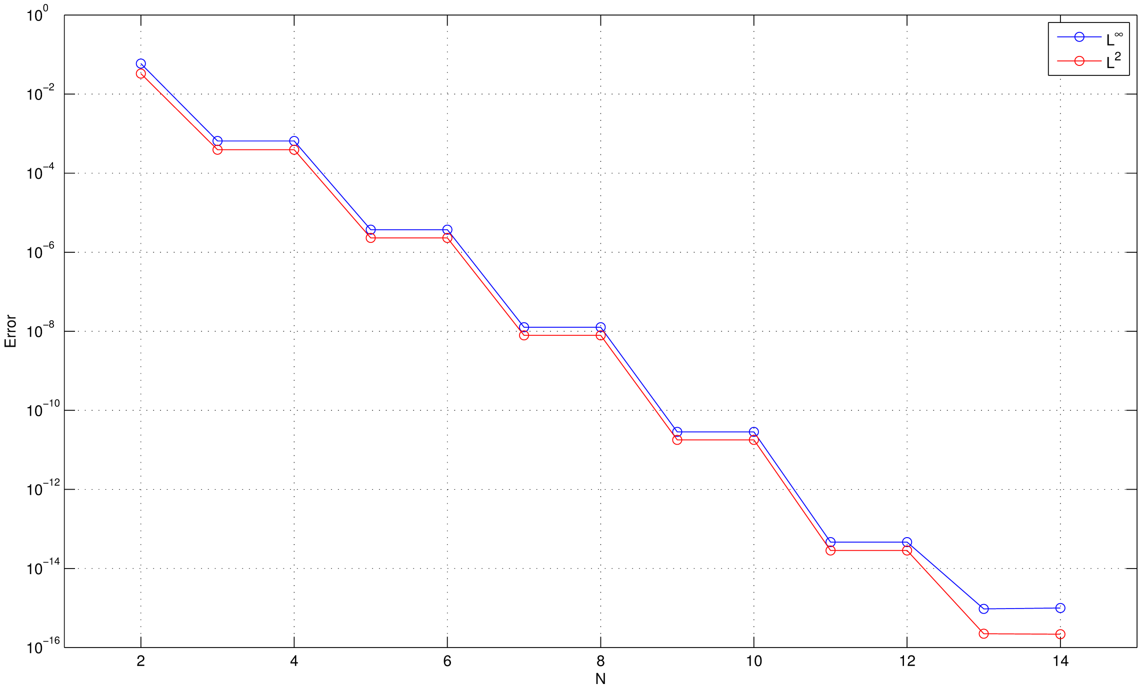


Figure 3.2: Convergence for subproblem (3.1f).

`L-2 convergence exponential with gamma:  1.7698 and delta:  1.1627.`
Note that the parameter $0 < \delta < \infty$ (high values are good) is more important than γ (which is often omitted). The plots looks like a *staircase*, this comes from the fact, that this particular exact solution uses only every second function in the space $V_{N,0}$.

Listing 3.3: Convergence estimates for (3.1f)

```
nvals = 2:14;
M = 10^5;
sigma = @(x) ones(size(x))./cosh(sin(pi*x));
f = @(x) pi^2*sin(pi*x);
exact = @(x) sinh(sin(pi*x));

xpts = linspace(0,1,M+1)';
xpts = xpts(2:end-1);

l2err = [];
lierr = [];

for N = nvals
    A = getGalMat(sigma, N);
    L = getrhsvector(f, N);
    mu = A \ L;

    approx = evaltrigsum(mu, M);
    err = abs(approx - exact(xpts));

```

```
    l2err = [l2err; sqrt(sum(err.^2)/M)];
    lierr = [lierr; max(err)];
end

semilogy(nvals, lierr, 'bo-');
hold on; grid on;
plot(nvals, l2err, 'ro-');
xlim([1, 15]);
xlabel('N');
ylabel('Error');
legend('L^\infty', 'L^2');

P = polyfit(log(nvals), log(log(lierr')),1);
gamma_inf=-real(exp(P(2)));
delta_inf=P(1);

disp(['L-inf convergence exponential with gamma: '
   num2str(gamma_inf) ' and delta: ' num2str(delta_inf) ]);
Q = polyfit(log(nvals), log(log(l2err')), 1);
gamma_2=-real(exp(Q(2)));
delta_2=Q(1);
disp(['L-2 convergence exponential with gamma: '
   num2str(gamma_2) ' and delta: ' num2str(delta_2) ]);

plot(nvals, exp(-gamma_inf.*nvals.^delta_inf));
hold off;
```

(3.1g) Carry out the investigations requested in subproblem (3.1f) for

$$\sigma(x) = \begin{cases} 2 & \text{for } |x - \frac{1}{2}| < \frac{1}{4}, \\ 1 & \text{elsewhere,} \end{cases} \qquad f \equiv 1 \,,$$

this time using $N = 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024$. What kind of convergence do you observe? Relate with the observation made in subproblem (3.1f) and try to explain.

HINT: The exact solution is

$$u(x) = \begin{cases} \frac{3}{64} + \frac{1}{4}x - \frac{1}{4}x^2 & |x - \frac{1}{2}| < \frac{1}{4}, \\ \frac{1}{2}x - \frac{1}{2}x^2 & \text{elsewhere.} \end{cases}$$

Also recall the observations made in [NPDE, Exp. 1.6.31].

Solution:

See Figure 3.3 for the convergence plots and Listing 3.4 for the code used. The convergence is algebraic with rates 1 for the L^∞-norm and 1.25 for the L^2-norm.

Listing 3.4: Convergence estimates for subproblem (3.1f)

```
nvals = 2.^(1:10);
```

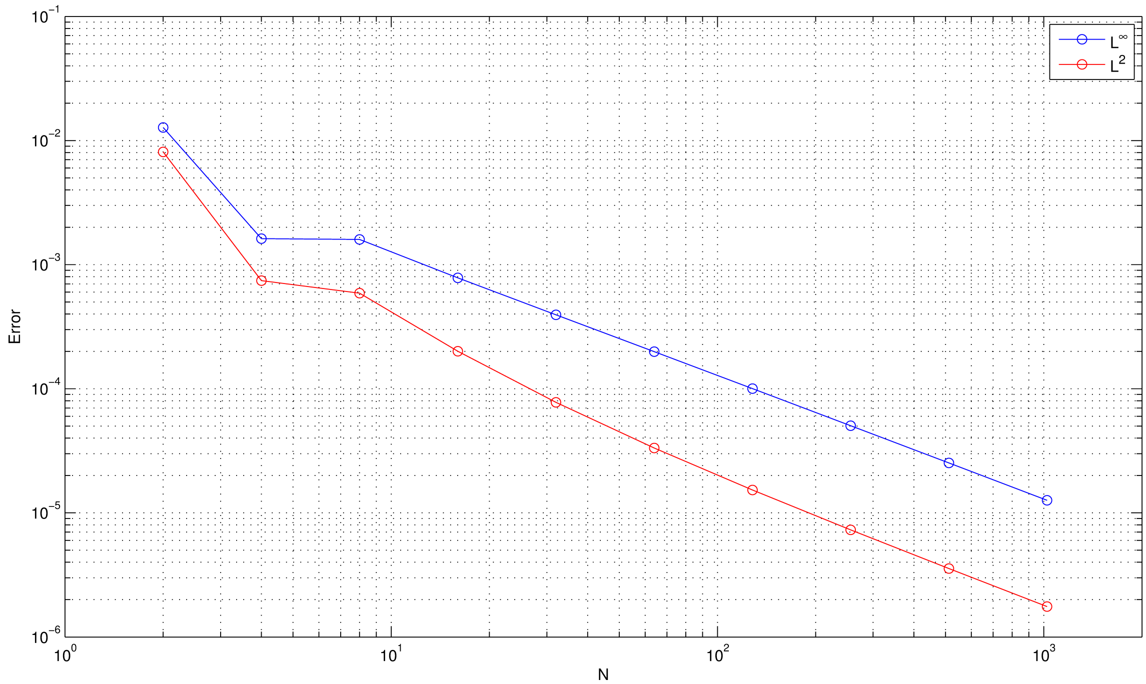


Figure 3.3: Convergence for subproblem (3.1f).

```
M = 10^5;
inside = @(x) abs(x-0.5) < 0.25;
outside = @(x) 1 - inside(x);
sigma = @(x) (1+inside(x)).*ones(size(x));
f = @(x) ones(size(x));
temp = @(x) 0.25*x - 0.25*x.^2;
exact = @(x) (1+outside(x)).*temp(x) +
   inside(x).*ones(size(x))*3/64;

xpts = linspace(0,1,M+1)';
xpts = xpts(2:end-1);

l2err = [];
lierr = [];

for N = nvals
    A = getGalMat(sigma, N);
    L = getrhsvector(f, N);
    mu = A \ L;

    approx = [evaltrigsum(mu, M)];
    err = abs(approx - exact(xpts));

    l2err = [l2err; sqrt(sum(err.^2)/M)];
    lierr = [lierr; max(err)];
end
```

```
loglog(nvals, lierr, 'bo-');
hold on; grid on;
plot(nvals, l2err, 'ro-');
xlim([1, 2000]);
xlabel('N');
ylabel('Error');
legend('L^\infty', 'L^2');

P = polyfit(log(nvals), log(lierr'), 1);
disp(['L-inf convergence algebraic with rate: '
    num2str(-P(1))]);
P = polyfit(log(nvals), log(l2err'), 1);
disp(['L-2 convergence algebraic with rate: '
    num2str(-P(1))]);
```

Listing 3.5: Testcalls for Problem 3.1

```
M = 6;

sigma = @(x)(1./(cosh(sin(pi*x))));
N=5;
fprintf('\n\n##getGalMat:')
getGalMat(sigma,N)

f = @(x) (pi^2 * sin(pi*x));
fprintf('\n\n##getrhsvector:')
getrhsvector(f,N)
```

Listing 3.6: Output for Testcalls for Problem 3.1

```
test_call2

##getGalMat:
ans =

    4.4209    0.0000    1.4066    0.0000    0.2058
    0.0000   16.1117    0.0000    3.4734   -0.0000
    1.4066    0.0000   35.9390    0.0000    6.4682
    0.0000    3.4734    0.0000   63.8443   -0.0000
    0.2058   -0.0000    6.4682   -0.0000   99.7504

##getrhsvector:
ans =

    4.9348
    0.0000
    0.0000
   -0.0000
    0.0000
```

Problem 3.2 Linear Finite Elements for the Brachistochrone Problem

In this problem we focus on the Galerkin discretization of the variational formulation for the brachistochrone problem by means of linear finite elements as introduced in [NPDE, Section 1.5.2.2].

We remind that the variational problem reads as: Find $\mathbf{u} \in V$ so that

$$\int_0^1 \left(\frac{\mathbf{u}' \cdot \mathbf{v}'}{\sqrt{-u_2}\|\mathbf{u}'\|} - \frac{v_2\|\mathbf{u}'\|}{2\sqrt{-u_2}u_2} \right) \mathrm{d}\xi = 0 \quad \text{for all } \mathbf{v} \in V_0, \tag{3.2.1}$$

$V_0 := \left\{ \mathbf{v} \in (\mathcal{C}^1_{\mathrm{pw}}([0,1]))^2 \,\middle|\, \mathbf{v}(0) = \mathbf{v}(1) = \mathbf{0} \right\}$ and $V := \left\{ \mathbf{v} \in (\mathcal{C}^1_{\mathrm{pw}}([0,1]))^2 \,\middle|\, \mathbf{v}(0) = \boldsymbol{a},\ \mathbf{v}(1) = \boldsymbol{b} \right\}$. Throughout we use equidistant meshes $\mathcal{M} = \{]x_{j-1} := \frac{j-1}{M}, x_j := \frac{j}{M}[, j = 1, \dots, M\}$ of $[0,1]$ and the standard "tent function" basis $\mathfrak{B}$ of the Galerkin trial space

$$V_{N,0} = (\mathcal{S}^0_{1,0}(\mathcal{M}))^2 = \left\{ \begin{array}{l} \mathbf{v} \in (C^0([0,1]))^2 : \ \mathbf{v}_{|[x_{i-1},x_i]} \text{ linear }, \\ i = 1, \dots, M,\ \mathbf{v}(0) = \mathbf{v}(1) = 0 \end{array} \right\}. \tag{3.2.2}$$

This is explained in detail in [NPDE, § 1.5.82], see, in particular, [NPDE, Eq. (1.5.83)]. It is recommended to use the ordering of the basis functions implied by [NPDE, Eq. (1.5.83)], though you are free to use any other scheme.

(3.2a) Determine a "$\mathbf{u}$-dependent coefficient function" $\sigma(\xi) = \sigma(\mathbf{u})(\xi)$ and a "$\mathbf{u}$-dependent source function" $\mathbf{f}(\xi) = \mathbf{f}(\mathbf{u})(\xi)$ such that the variational formulation (3.2.1) can be written as

$$\mathbf{u} \in V: \quad \int_0^1 \sigma(\mathbf{u})(\xi)\mathbf{u}'(\xi) \cdot \mathbf{v}'(\xi)\,\mathrm{d}\xi = \int_0^1 \mathbf{f}(\mathbf{u})(\xi) \cdot \mathbf{v}(\xi)\,\mathrm{d}\xi \quad \forall \mathbf{v} \in V_0\,. \tag{3.2.3}$$

The point of recasting (3.2.1) in this form is the reduction to the structure of a linear variational problem. For the elastic string model this has proved highly useful as regards the implementation of Galerkin discretizations in [NPDE, § 1.5.55], Code [NPDE, Code 1.5.58], and [NPDE, § 1.5.82], Code [NPDE, Code 1.5.92]. Please study the latter example and code again, in case you do not remember the rationale behind (3.2.3).

Solution: We have:

$$\sigma(\mathbf{u}) = \frac{1}{\sqrt{-u_2}\|\mathbf{u}'\|}, \quad \mathbf{f}(\mathbf{u}) = \frac{\|\mathbf{u}'\|}{2\sqrt{-u_2}u_2} \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

(3.2b) Implement a MATLAB function

```
s = sigma(mu, xi)
```

where:

- `mu` is a $2 \times (M+1)$ matrix containing the components $\{\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_{M+1}\}$ of $\mathbf{u}_N$ with respect to the basis representation of the curve (where $\boldsymbol{\mu}_1 = \mathbf{u}(0)$ and $\boldsymbol{\mu}_{M+1} = \mathbf{u}(1)$ are the pinning points);
- `xi` is a vector of evaluation points in the interval $]0,1[$.

The output is a vector `s` with the evaluations of the scalar function $\sigma = \sigma(\mathbf{u})$ at the mesh points `xi`.

HINT: You may use the MATLAB function `linterp` to get a piecewise linear interpolation of $\mathbf{u}_N$ at the evaluation points.
Remember that the mesh on which you have the coefficients `mu` is equispaced.
The derivative of $\boldsymbol{u}_N$ is piecewise constant. You don't have to worry if an evaluation point is also a mesh point, where the derivative is discontinuous; in this case, you can take either the left or the right derivative.
A reference implementation `sigma_ref` is available in the file `sigma_ref.p`.

Solution: See listing 3.7 for the code.

Listing 3.7: Implementation for `sigma`

```
function s = sigma(mu,xi)

    mesh = linspace(0,1,length(mu));
    h = mesh(2)-mesh(1);
    d = (mu(:,2:end)-mu(:,1:end-1))/h; % derivative of u
    dnorm = sqrt(d(1,:).^2 + d(2,:).^2); % norm of the derivative
    mu_xi = linterp(mesh,mu(2,:),xi); % interpolate the second component in the evaluation points

    for j=1:length(xi)
     index = find(mesh>xi(j),1);     % find the index of the first mesh point greater than xi(j)
                                     % the derivative in xi corresponds to the entry d(index-1)
                                     % at mesh points, we consider the right derivative
     s(j) = 1/(sqrt(-mu_xi(j))*dnorm(index-1));
    end
```

(3.2c) Write a MATLAB function

```
f = sourcefn(mu, xi)
```

where the input arguments are the same as in subproblem (3.2b) and the output is a $2 \times K$ matrix, with K the length of `xi`, containing the evaluations of the $\mathbf{u}$-dependent source function $\mathbf{f} = \mathbf{f}(\mathbf{u})$ at the mesh points `xi`.
If an evaluation point concides with a mesh point, where the derivative of $\boldsymbol{u}_N$ is not uniquely defined, consider either the left or the right derivative.

Solution: See listing 3.8 for the code. operations as far as possible.

Listing 3.8: Implementation for `sourcefn`

```
function f = sourcefn(mu,xi)

    M =length(mu)-1;
    mesh = linspace(0,1,length(mu));
    h = mesh(2)-mesh(1);
    d = (mu(:,2:end)-mu(:,1:end-1))/h; % derivative of u
    dnorm = sqrt(d(1,:).^2 + d(2,:).^2); % norm of the derivative
    mu_xi = linterp(mesh,mu(2,:),xi);

    f(1,:)= zeros(1,length(xi));

    for j=1:length(xi)
     index = find(mesh>xi(j),1);  % find the index of the first mesh point greater than xi(j)
                                  % the derivative in xi corresponds to the entry d(index-1)
                                  % at mesh points, we consider the right derivative
     f(2,j) = dnorm(index-1)/(2*sqrt(-mu_xi(j))*mu_xi(j));
    end
```

(3.2d) Implement a MATLAB function

`t = traveltime (mu)`

which accepts as input the vector `mu` as in (3.2b), and returns the approximate value of the functional

$$J(\mathbf{u}_N) = \int_0^1 \frac{\|\mathbf{u}_N'(\xi)\|}{\sqrt{-(\mathbf{u}_N)_2(\xi)}}\,\mathrm{d}\xi \qquad (3.2.4)$$

representing the time needed to go from $\mathbf{u}(0)$ to $\mathbf{u}(1)$ along the curve $\mathbf{u}_N$.

For the computation of the integral in (3.2.4), use the midpoint rule (see [NPDE, Eq. (1.5.77)]). Note that the trapezoidal rule would be inappropriate because of the singularity of the functional at the origin.

HINT: A reference implementation `traveltime_ref` is available in the file `traveltime_ref.p`.

Solution: See listing 3.9 for the code.

Listing 3.9: Implementation for `traveltime`

```
function t = traveltime(mu)

```

```
    mesh = linspace(0,1,length(mu));
    h = mesh(2)-mesh(1);
    d = (mu(:,2:end)-mu(:,1:end-1))/h;
    dnorm = sqrt(d(1,:).^2 + d(2,:).^2);
    % TRAPEZOIDAL RULE
%     mu2 = mu(2,:);
%     t =
   sum((dnorm(1:end-1)+dnorm(2:end))./(sqrt(-mu2(2:(end-1)))))+...
%           ...+dnorm(end)/sqrt(-mu2(end));
%     if abs(mu2(1))>eps
%         t = t+dnorm(1)/sqrt(-mu2(1));
%     end
    % MIDPOINT RULE
    xi = h/2:h:(1-h/2);
    mu_xi = linterp(mesh,mu(2,:),xi);
    t = sum(dnorm./(sqrt(-mu_xi)))*h;
```

(3.2e) Proceeding as in [NPDE, § 1.5.82], the discretization of (3.2.3) leads to a *nonlinear* system of equations of the form

$$\begin{pmatrix} \mathbf{R}(\vec{\mu}) & 0 \\ 0 & \mathbf{R}(\vec{\mu}) \end{pmatrix} \vec{\mu} = \begin{pmatrix} \vec{\varphi}_1(\vec{\mu}) \\ \vec{\varphi}_2(\vec{\mu}) \end{pmatrix}, \tag{3.2.5}$$

with $\mathbf{R}(\vec{\mu}) \in \mathbb{R}^{M-1,M-1}$ and $\vec{\varphi}_i(\vec{\mu}) \in \mathbb{R}^{M-1}$. Write a MATLAB function

```
R = Rmat(mu)
```

such that, given the coefficients $\vec{\mu}$ = mu in input (as in subproblem (3.2a)), returns the matrix R = $\mathbf{R}(\vec{\mu})$.
For the evaluation of the integrals, use the midpoint rule [NPDE, Eq. (1.5.77)].

HINT: Compute the matrix including also the rows and columns referring to the two basis functions for the offset function. In this way, your matrix R will have dimensions $(M+1)\times(M+1)$. Then, in subproblem (3.2g), where you will have to solve the linear system, you have to consider just the entries relative to the inner nodes (i.e. you have to exclude the first and last columns and rows of R). The reason for doing this is that with such R it will be easier, in subproblem (3.2g), to modify the right hand side to take into account the boundary conditions.
A reference implementation R_ref is available in the file R_ref.p.

Solution: See listing 3.10 for the code.

Listing 3.10: Implementation for Rmat

```
function R = Rmat(mu)

    mesh = linspace(0,1,length(mu));
    h = mesh(2)-mesh(1);
    xi = h/2:h:(1-h/2); % midpoints, where we will evaluate
       sigma
```

```
    M = length(mu)-1;
    % Computation of the r_j
    s = sigma(mu,xi);
    r = s./h;

    % Assemble tridiagonal matrix R
    R = gallery('tridiag',[-r(1), -r(2:M-1),-r(M)],[r(1),
       r(1:M-1)+r(2:M), r(M)],[-r(1), -r(2:M-1), -r(M)]);
```

(3.2f) Write a MATLAB function

`phi = rhs(mu)`

which, given the coefficients `mu` in input, returns as output the right hand side vector from (3.2.5)

$$\vec{\varphi}(\vec{\mu}) = \begin{pmatrix} \vec{\varphi}_1(\vec{\mu}) \\ \vec{\varphi}_2(\vec{\mu}) \end{pmatrix} \in \mathbb{R}^{2M-2} . \tag{3.2.6}$$

For integration, consider the composite trapezoidal quadrature rule [NPDE, Eq. (1.5.72)]. At the origin (where the source function is singular), consider the integrand to be zero.
HINT: A reference implementation `rhs_ref` is available in the file `rhs_ref.p`.

Solution: See listing 3.11 for the code.

Listing 3.11: Implementation for `rhs`

```
function phi = rhs(mu)

    mesh = linspace(0,1,length(mu));
    h = mesh(2)-mesh(1);
    % TRAPEZOIDAL RULE
    xxi = h:h:(1-h);
    phi =
       (sourcefn(mu,xxi(1:end))+sourcefn(mu,xxi(1:end)))*h/2;
%     MIDPOINT RULE: DOESN'T WORK!
%     xi = h/2:h:(1-h/2);
%     sourcefn(mu,xxi); % for debugging
%     2*sourcefn(mu,xxi) % for debugging
%     source = sourcefn(mu,xi);
%     source(:,1:end-1) + source(:,2:end) % for debugging
%     phi = source(:,1:end-1)*h/2 + source(:,2:end)*h/2;
```

(3.2g) The nonlinear system (3.2.6) can be solved by *fixed point iteration*. In this way, an approximate solution is computed solving, at each iteration, a *linear* system of equations (see [NPDE, § 1.5.82], [NPDE, Code 1.5.92]).
Implement a MATLAB function

`mufinal = solvebrachlin(mu0, tol)`

to compute the approximate solution (i.e. the coefficients `mufinal` with respect to the basis functions) using the fixed point iteration.
The input arguments are the initial guess `mu0` and the tolerance `tol` for the relative error in the fixed point iteration algorithm (see [NPDE, Code 1.5.92], line 41).
For each iteration, plot the shape of the curve (see [NPDE, Code 1.5.92], lines 19-22).
Plot the travel time as defined in subproblem (3.2d) with respect to the number of iteration steps.
Remark: Remember to take into account the boundary conditions.

HINT: The solution should look like the cycloid

$$\mathbf{u}(\xi) = \begin{pmatrix} \pi\xi - \sin(\pi\xi) \\ \cos(\pi\xi) - 1 \end{pmatrix}, \quad 0 \leq \xi \leq 1\,, \tag{3.2.7}$$

that was shown in the previous assignment to be a strong solution.
A reference implementation `solvebrachlin_ref` is available in the file `solvebrachlin_ref.p`.

Solution: See listing 3.12 for the code.

Listing 3.12: Implementation for `solvebrachlin`

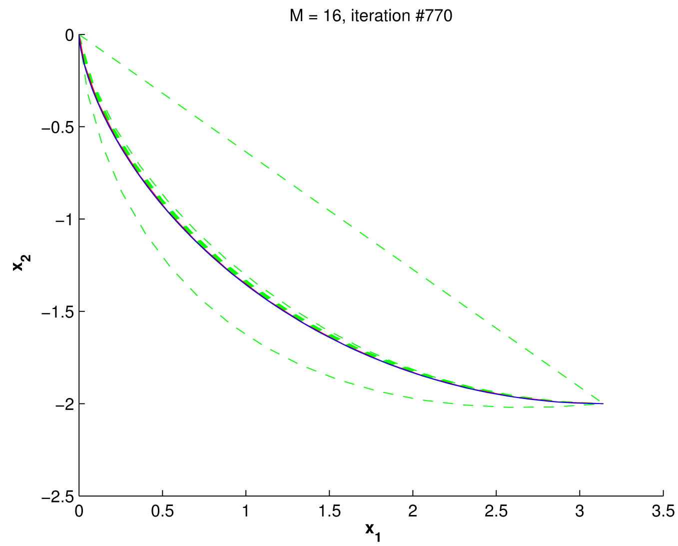
```
function [mu_new,figsol] = solvebrachlin(init_guess,tol)
% M intervals, M-1 interior nodes
M = length(init_guess)-1;
h= 1/M; %meshwidth

figsol = figure; hold on;
maxiter=10000;
mu_new = init_guess;
t = [];
for k=1:maxiter
    mu = mu_new;
    % Plot shape of string
    plot(mu(1,:), mu(2,:),'--g'); drawnow;
    title(sprintf('M = %d, iteration #%d', M,k));
    xlabel('{\bf x_1}'); ylabel('{\bf x_2}');
    t = [t; traveltime(mu)];

    R = Rmat(mu);
    % Computation of right hand side
    phi = rhs(mu);
    phi1 = phi(1,:);
    phi2 = phi(2,:);

    % Modify right hand side to take in account to the offset function
    phi1(1)= phi1(1) - R(2,1)*mu(1,1);
    phi1(M-1) = phi1(M-1) -R(M,M+1)*mu(1,end);
    phi2(1) = phi2(1) - R(2,1)*mu(2,1);
    phi2(M-1) = phi2(M-1) -R(M,M+1)*mu(2,end);

```




```
    % Solve linear system and compute new iterate
    mu_new = [mu(:,1),[(R((2:end-1),(2:end-1))\phi1')';
                (R((2:end-1),(2:end-1))\phi2')'],mu(:,end)];
    % Check simple termination criterion for fixed point iteration.
    if (norm(mu_new - mu,'fro') < tol*norm(mu_new,'fro')/M)
        plot(mu_new(1,:),mu_new(2,:),'r');
        xi = linspace(0,1);
        plot(pi*xi-sin(pi*xi),cos(pi*xi)-1,'b');
    break;
    end

end
    figure
    plot(t);
```

The solution for M=16, with straight line as initial guess, should look like the following:

(3.2h) The fixed point iteration algorithm converges quite slowly to the approximate solution. To improve this, we use *nested iterations*:

a) start from an initial guess `mu0` and an initial *coarse* mesh `mesh0` and compute the solution `mu1`;

b) consider the mesh `mu1` obtained from `mu0` inserting the midpoints of the interval as mesh points (thus doubling the number of mesh points) and compute the solution `mu2` considering `mu1` as initial guess;

c) repeat point b) iteratively until the desired final meshwidth level `L` is reached.

In point b), to get the initial guess, one has to extend a piecewise linear function defined on a coarser grid to a finer nested grid. To achieve this, piecewise linear interpolation can be used.

Remark: The advantage of considering the relative error tolerance for the fixed point iteration adapted to the meshsize (see [NPDE, Code 1.5.92], line 41) is that in all iterations from point b) the same tolerance `tol` can be used.

Write a MATLAB function

```
mufinal = nestitbrachlin(uend,L,tol)
```

to solve the brachistochrone problem using nested iterations.
Here, `uend` is the right pinning point, while the left pinning point is consider to be the origin. `L` is the number of refinement levels. Start from a mesh of `M=2` intervals, corresponding to the level `L=0`.
HINT: For the linear interpolation for the initial guess in point b), use the MATLAB function `linterp`.

Solution: See listing 3.13 for the code.

Listing 3.13: Implementation for `nestitbrachlin`

```
function mufinal = nestitbrachlin(uend,L,tol)

u0 = [0;0];
mu = [u0, (u0+uend)/2, uend];
level=0;

while(level<L+1)
    mu_new = solvebrachlin(mu,tol);
    meshsize = 1/(2^(level+1));
    level = level+1;
    clear mu;
    mu(1,:) = linterp(0:meshsize:1,mu_new(1,:),0:meshsize/2:1);
    mu(2,:) = linterp(0:meshsize:1,mu_new(2,:),0:meshsize/2:1);
end
mufinal(1,:) = mu(1,:);
mufinal(2,:) = mu(2,:);
```

Listing 3.14: Testcalls for Problem 3.2

```
u0 = [0;0];
u1 = [pi;-2];
M = 4;
mesh = linspace(0,1,M+1);
h = mesh(2)-mesh(1);
mu = u0*(1-(0:1/M:1))+u1*(0:1/M:1);

fprintf('\n\n##sigma:')
xi = h/2:h:(1-h/2);
sigma(mu, xi)
```

```

fprintf('\n\n##sourcefn:')
xxi = h:h:(1-h);
sourcefn(mu,xxi)

fprintf('\n\n#traveltime:')
traveltime(mu)

fprintf('\n\n##Rmat:')
Rmat(mu)

fprintf('\n\n##rhs:')
rhs(mu)

fprintf('\n\n##solvebrachlin:')
solvebrachlin(mu,10^(-5))

fprintf('\n\n##nestitbrachlin:')
nestitbrachlin([pi;-2],3,10^(-5))
```

Listing 3.15: Output for Testcalls for Problem 3.2

```
>> test_call_fem

##sigma:
ans =

    0.5370    0.3101    0.2402    0.2030

##sourcefn:
ans =

         0         0         0
   -5.2668   -1.8621   -1.0136

#traveltime:
ans =

    4.4737

##Rmat:
ans =

   (1,1)        2.1481
   (2,1)       -2.1481
   (1,2)       -2.1481
   (2,2)        3.3883
   (3,2)       -1.2402
   (2,3)       -1.2402
   (3,3)        2.2009
```

```
  (4,3)      -0.9607
  (3,4)      -0.9607
  (4,4)       1.7726
  (5,4)      -0.8119
  (4,5)      -0.8119
  (5,5)       0.8119

##rhs:
ans =

        0         0         0
   -1.3167   -0.4655   -0.2534

##solvebrachlin:
ans =

        0    0.3114    1.0169    1.9977    3.1416
        0   -0.6794   -1.3570   -1.8269   -2.0000

##nestitbrachlin:
ans =

  Columns 1 through 9

        0    0.0510    0.1019    0.2179    0.3339    0.4994
      0.6650    0.8683    1.0717
        0   -0.1679   -0.3358   -0.5287   -0.7215   -0.9031
     -1.0847   -1.2425   -1.4002

  Columns 10 through 17

   1.3040    1.5364    1.7906    2.0447    2.3145    2.5843
      2.8629    3.1416
  -1.5280   -1.6558   -1.7500   -1.8442   -1.9022   -1.9602
     -1.9801   -2.0000
```

Problem 3.3 Spline Collocation Method

[NPDE, Section 1.5.3.2] introduces the spline collocation method for 2-point boundary value problems based on natural cubic splines, see [NPDE, Def. 1.5.116]. In this problem we consider the simple BVP

$$-\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f(x) \quad \text{in } [0,1], \quad u(0) = u(1) = 0, \tag{3.3.1}$$

$f \in C^0([0,1])$, and its cubic spline collocation discretization based on the knot set

$$\mathcal{T} := \{jh\}_{j=0}^{M}, \quad h := 1/M, \quad M \in \mathbb{N}, \tag{3.3.2}$$

and on the set of collocation nodes $\{jh\}_{j=1}^{M-1}$.

(3.3a) Refresh your knowledge of the basic idea of discretization by collocation as explained in the beginning of [NPDE, Section 1.5.3].

(3.3b) Any cubic spline $s \in \mathcal{S}_{3,\mathcal{T}}$ has the local monomial representation

$$s(x) = a_i(x - x_{i-1})^3 + b_i(x - x_{i-1})^2 + c_i(x - x_{i-1}) + d_i, \quad x_{i-1} < x < x_i,\ i = 1, \dots, M, \tag{3.3.3}$$

with coefficients $a_i, b_i, c_i, d_i \in \mathbb{R}$. This local monomial representation will be used in the sequel.

Denote $f_i := f(x_i)$, $i = 0, \dots, M$.

Show that the collocation conditions [NPDE, Eq. (1.5.100)] imply the following formulas

$$a_i = \frac{f_{i-1} - f_i}{6h}, \quad i = 1, \dots, M, \tag{3.3.4a}$$

$$b_1 = 0 \tag{3.3.4b}$$

$$b_i = -\tfrac{1}{2} f_{i-1}, \quad i = 2, \dots, M, \tag{3.3.4c}$$

for the coefficients of the local monomial representation of the spline solving the collocation equations (in the fist equation we have set $f_M := s''(1) = 0$).

Solution: A natural cubic spline $s : [0, 1] \to \mathbb{R}$ satisfies

(i) $s \in \mathcal{C}^2([0, 1])$,

(ii) $s\big|_{[x_{j-1}, x_j]} \in \mathcal{P}_3(\mathbb{R})$,

(iii) $s''(0) = s''(1) = 0$.

With (ii) we know that for $x_{i-1} \leq x \leq x_i$, $i = 1, \dots, M$ we have

$$\begin{aligned} s(x) &= a_i(x - x_{i-1})^3 + b_i(x - x_{i-1})^2 + c_i(x - x_{i-1}) + d_i, \\ s'(x) &= 3a_i(x - x_{i-1})^2 + 2b_i(x - x_{i-1}) + c_i, \\ s''(x) &= 6a_i(x - x_{i-1}) + 2b_i. \end{aligned}$$

Since $-s''(x_{i-1}) = f(x_{i-1})$ we have .

$$2b_i = -f(x_{i-1}) = -f_{i-1} \quad \Rightarrow \quad b_i = \frac{1}{2} f_{i-1}, \quad i = 2, \dots, M,$$

and $b_1 = 0$ since $s''(0) = 0$. We also have for $-s''(x_i) = f(x_i)$ that

$$-6a_i h - f_{i-1} = f_i \quad \Rightarrow \quad a_i = \frac{f_{i-1} - f_i}{6h} \quad i = 1, \dots, M.$$

(3.3c) The equations (3.3.4) are too few to determine all unknown coefficients. To obtain further equations, use the continuity conditions for the "zeroth" and first derivative of a cubic spline, the collocation and boundary conditions at $x = 0$ and $x = 1$, and the relationships (3.3.4) to establish

$$d_1 = 0 \tag{3.3.5a}$$

$$d_{i+1} - 2d_i + d_{i-1} = -\frac{h^2}{6}(f_i + 4f_{i-1} + f_{i-2}), \qquad 2 \leq i \leq M - 1, \tag{3.3.5b}$$

$$c_2 - c_1 = -\frac{h}{2} f_1, \tag{3.3.5c}$$

$$c_{i+1} - c_i = -\frac{h}{2}(f_i + f_{i-1}), \qquad 2 \leq i \leq M - 1, \tag{3.3.5d}$$

$$c_M h + d_M = \frac{h^2}{3} f_{M-1}, \tag{3.3.5e}$$

$$c_{M-1} h + d_{M-1} - d_M = \frac{h^2}{6}(2f_{M-2} + f_{M-1}). \tag{3.3.5f}$$

HINT: Recall the formulas for interpolating natural cubic splines from [NCSE, Sect. 3.8.1].

Solution: Since $s'(x)$ is continuous we have

$$\begin{aligned} c_{i+1} = s'(x_i) &= 3a_i h^2 + 2b_i h + c_i \\ &= 3\,\frac{f_{i-1} - f_i}{6h} h^2 - 2\,\frac{f_{i-1}}{2} h + c_i \\ &= -\frac{f_i + f_{i-1}}{2} h + c_i. \end{aligned}$$

Therefore $c_{i+1} - c_i = -\frac{f_i + f_{i-1}}{2} h$ for $i = 2, \dots, M-1$ and $c_{i+1} - c_i = -\frac{f_i}{2} h$ for $i = 1$.

Furthermore, since $s(x)$ is continuous, we obtain

$$\begin{aligned} & d_i = s(x_i) = a_{i-1} h^3 + b_{i-1} h^2 + c_{i-1} h + d_{i-1}, \\ \Rightarrow \quad & d_{i+1} - d_i = a_i h^3 + b_i h^2 + c_i h; \end{aligned}$$

this implies

$$\begin{aligned} \Rightarrow \quad d_{i+1} - 2d_i + d_{i-1} \; &= h^3 \left(\frac{f_{i-1} - f_i - f_{i-2} + f_{i-1}}{6h} \right) + h^2 \left(\frac{-f_{i-1} + f_{i-2}}{2} \right) + \frac{h^2}{2}(-f_{i-1} - f_{i-2}) = \\ &= -\frac{h^2}{6}(f_i + 4f_{i-1} + f_{i-2}), \end{aligned}$$

for $i = 2, \dots, M-1$, and

$$c_{M-1} h + d_{M-1} - d_M = -a_{M-1} h^3 - b_{M-1} h^2 = \frac{h^2}{6}(2f_{M-2} + f_{M-1}).$$

Additionally, the boundary conditions $s(0) = s(1)$ imply, respectively, $d_1 = 0$ and

$$c_M h + d_M = \frac{h^2}{3} f_{M-1}.$$

(3.3d) Write a MATLAB function

```
function [a,b,c,d] = natcubsplinecoll(M,f)
```

that computes the local monomial expansion coefficients of the natural cubic spline collocation solution of (3.3.1) with node set (3.3.2). These coefficients are to be returned in the row vectors `a`, `b`, `c`, `d` of length M. The argument `f` is a handle to the (continuous) right hand side function f.

Solution: See Listing 3.16.

Listing 3.16: Code for subproblem (3.3d).

```
function [a,b,c,d] = natcubsplinecoll(M,f)

% Computes the coefficients in the monomial expansion of the spline
   for
% equispace knots
% INPUT:
% M = number of intervals
% f = FHandle to right-hand side
% OUTPUT:
```

```
% column vectors of coefficients in the monomial expansion

h = 1/M;
x = linspace(0,1,M+1);
fval = f(x);
fval = fval(:);
a = (fval(1:end-1)-fval(2:end))/(6*h);
b = [0; -fval(2:end-1)/2];
A11 = diag([-ones(M-1,1);h])+diag(ones(M-1,1),1);
A12 = zeros(M,M);
A12(end,end) = 1;
A21 = zeros(M,M);
A21(end,end-1) = h;
A22 =
    diag([1;-2*ones(M-2,1);-1])+diag([0;ones(M-2,1)],1)+diag([ones(M-2,1);1],-1);
A = [A11 A12; A21 A22];
RHS = zeros(2*M,1);
RHS(1:M-1)=(fval(2:end-1)+fval(1:end-2))*(-h/2);
RHS(M) = fval(end-1)*h^2/3;
RHS(M+1) = 0;
RHS(M+2:end-1) = -h^2/6*(fval(3:M)+4*fval(2:M-1)+fval(1:M-2));
RHS(end) = h^2/6*(2*fval(M-1)+fval(M));
res = A\RHS;
c = res(1:M);
d = res(M+1:end);
```

(3.3e) Plot the approximate solution of (3.3.1) with $f(x) = \sin(2\pi x)$ obtained by natural cubic spline collocation on an equidistant node set (3.3.1) with $M = 5, 10, 20$.

Solution: See Listing 3.17.

Listing 3.17: Code for subproblem (3.3e).

```
f = @(x) sin(2*pi.*x(:));

M = 20;
[a, b, c, d]=natcubsplinecoll(M,f);
x = linspace(0,1,M+1);
xx = linspace(0,1,4*M+1);
yy = [];
for i=1:M
    yval = a(i)*(xx((i-1)*4+1:i*4+1)-x(i)).^3 +
        b(i)*(xx((i-1)*4+1:i*4+1)-x(i)).^2 +
        c(i)*(xx((i-1)*4+1:i*4+1)-x(i)) + d(i);
    yy = [yy(1:end-1) yval];
end

plot(xx,yy,'o')
```

For $M = 20$ we obtain the plot shown in Figure 3.4.

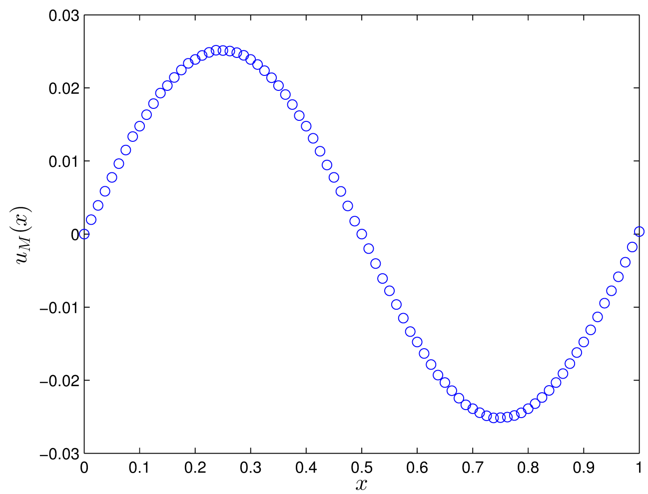


Figure 3.4: Spline collocation method for $M = 20$ mesh points.

(3.3f) As outlined in [NCSE, Rem. 9.1.4], investigate the convergence of the method on the same problem as in subproblem (3.3e) with $M = 5, 10, 20, 40, 80$ in the L^2-norm. In order to approximate the integral for computation of the norm use the composite 2-point Gauss quadrature rule.

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), see [NCSE, Rem. 10.1.3].

Solution:

Listing 3.18: Implementation for subproblem (3.3f)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SPLINECONV computes the convergence rate of a
%     solution of a model BVP problem
%     using the spline collocation method
%

% input data
M = 2.^(5:10)';                                  % number of nodes
f = @(x)  sin(2*pi.*x(:));                       % right-hand side
u_exact = @(x) 1/(4*pi^2)*sin(2*pi.*x(:));      % exact solution

% loop over M
l2error = zeros(length(M),1);
for i = 1:length(M)

    % mesh
    h = 1/M(i);
    x = linspace(0,1,M(i)+1)';

    % coefficents
    [a, b, c, d]=natcubsplinecoll(M(i),f);

    % compute quadrature points
    xpts = repmat(x(1:length(x)-1)+h/2,2,1)'; xpts = xpts(:);
```

```
    gqpts = repmat([-1;1]/sqrt(3),1,M(i))'*h/2; gqpts = gqpts(:) +
       xpts;
    gqpts = sort(gqpts);

    % compute solution in quadrature points
    u = [];
    for j=1:M(i)
        uval = a(j)*(gqpts((j-1)*2+1:j*2)-x(j)).^3 +
           b(j)*(gqpts((j-1)*2+1:j*2)-x(j)).^2 +
           c(j)*(gqpts((j-1)*2+1:j*2)-x(j)) + d(j);
        u = [u; uval];
    end

    % compute error
    l2error(i) = sqrt(sum(h/2*(u-u_exact(gqpts)).^2));
end

% compute convergence rate
p = polyfit(log(M),log(l2error),1);
s = p(1);
fprintf('Convergence rate s = %2.4f\n',abs(s));

% plot convergence rate
figure(1)
h = axes;
loglog(M,l2error,'bo-')
hold on;
add_Slope(gca,'NorthEast',p(1));
grid on
set(h,'FontSize',14);
xlabel('M')
ylabel('L^2-error')
%print -depsc splineconv.eps
```

We obtain the optimal convergence rate $s = 2$ as shown in Figure 3.5.

Listing 3.19: Testcalls for Problem 3.3

```

fprintf('\n\n##natcubsplinecoll:')

f = @(x) sin(2*pi.*x(:));
M = 5;
[a, b, c, d]=natcubsplinecoll(M,f)
```

Listing 3.20: Output for Testcalls for Problem 3.3

```
>> test_call

##natcubsplinecoll:
a =
```

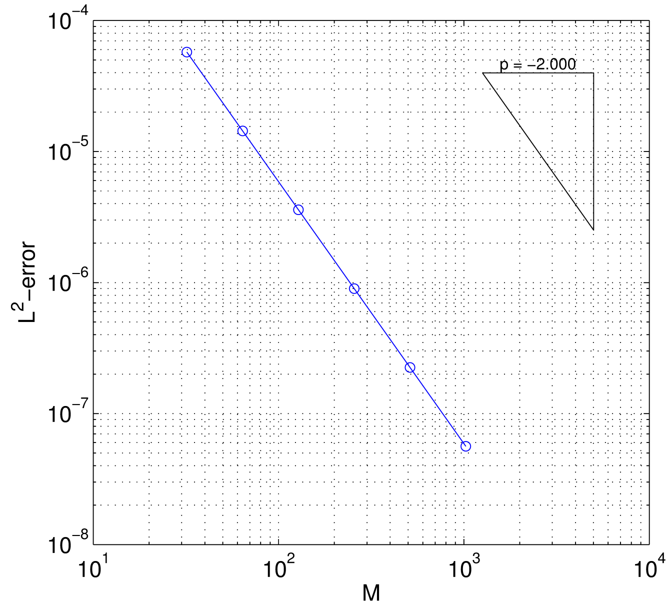


Figure 3.5: Convergence rate for the spline collocation method

```

   -0.7925
    0.3027
    0.9796
    0.3027
   -0.7925

b =

         0
   -0.4755
   -0.2939
    0.2939
    0.4755

c =

    0.1376
    0.0425
   -0.1114
   -0.1114
    0.0425

d =

         0
    0.0212
    0.0131
   -0.0131
```

```
   -0.0212
```

Published on March 04.
To be submitted on March 11.

References

[NPDE] Lecture Slides for the course “Numerical Methods for Partial Differential Equations”.SVN revision # 74075.

[1] M. Struwe. Analysis für Informatiker. Lecture notes, ETH Zürich, 2009. https://moodle-app1.net.ethz.ch/lms/mod/resource/index.php?id=145.

[NCSE] Lecture Slides for the course “Numerical Methods for CSE”.

Last modified on March 12, 2015