R. Hiptmair
L. Scarabosio
E. Fonn

Spring Term 2013

Numerical Methods for Partial Differential Equations

ETH Zürich
D-MATH

Homework Problem Sheet 2

Introduction. This problem sheet covers different approaches for the discretization of two-point boundary value problems as discussed in [NPDE, Sect. 1.5].

Problem 2.1 Linear Finite Elements in 1D (Core problem)

In [NPDE, Sect. 1.5.1.2] 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, see [NPDE, Eq. (1.5.84)]. In this problem we practise the crucial steps for the slightly modified linear variational problem

$$u \in \mathcal{C}^1_{\mathrm{pw},0}([a,b]): \quad \int_a^b \frac{\mathrm{d}u}{\mathrm{d}x}(x)\,\frac{\mathrm{d}v}{\mathrm{d}x}(x) + c\,u(x)v(x)\mathrm{d}x = \int_a^b g(x)v(x)\,\mathrm{d}x, \quad \forall v \in \mathcal{C}^1_{\mathrm{pw},0}([a,b]), \tag{2.1.1}$$

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.

(2.1a) Following [NPDE, Sect. 1.5.1.2] derive the Galerkin matrix for (2.1.1), when using the trial and test space $\mathcal{S}^0_{1,0}(\mathcal{M})$ 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, see [NPDE, Eq. (1.5.76)].

(2.1b) To obtain the right-hand side vector of the linear system arising from the Galerkin discretization of (2.1.1) as described in subproblem (2.1a), one relies on the trapezoidal rule [NPDE, Eq. (1.5.85)] on $\mathcal{M}$ for numerical quadrature.

Implement an efficient MATLAB 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.

(2.1c) State and justify the asymptotic computational complexity of `linfegalerkinsol` in terms of the problem size parameter N.

(2.1d) 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 $u(x) = \frac{1}{2}\sin(x)$.

(2.1e) 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 (2.1.1). This means that now we seek to solve (2.1.1) under the constraints $u(a) = u_a$, $u(b) = u_b$.
HINT: Use the offset function technique according to [NPDE, Rem. 1.5.90] and arrive at a modified right-hand side of the linear system of equations that incorporates the values u_a and u_b.

(2.1f) 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 $u(x) = \frac{1}{2}\cos(x)$.

(2.1g) Investigate the convergence [NPDE, Remark 1.6.38] of the finite element discretization developed in (2.1b) 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 (2.1d).

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.

(2.1h) Investigate the convergence [NPDE, Remark 1.6.38] in the L^2-norm

$$\|e\|_{L^2} := \left(\int_0^1 |e(x)|^2 \, \mathrm{d}x \right)^{\frac{1}{2}}, \quad e \in \mathcal{C}^0_{\mathrm{pw}}([0,1]). \tag{2.1.2}$$

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), see [NCSE, Rem. 10.1.3]. For $N = 10$, the error should be around 0.012.

(2.1i) Investigate the convergence [NPDE, Remark 1.6.38] in the energy seminorm

$$|e|_{H^1} := \left(\int_0^1 \left| \frac{\mathrm{d}e}{\mathrm{d}x}(x) \right|^2 \right)^{\frac{1}{2}}, \quad u \in \mathcal{C}^1_{\mathrm{pw}}([0,1]). \tag{2.1.3}$$

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.

Listing 2.1: Testcalls for Problem 2.1

```
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 2.2: Output for Testcalls for Problem 2.1

```
>> 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 2.2 Fourier Spectral Galerkin Scheme for Two-Point Boundary Value Problem

In [NPDE, Sect. 1.5.1.1] 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, Rem. 1.5.61].

Tis 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.

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

$$\int_0^1 \sigma(x)\frac{\mathrm{d}u}{\mathrm{d}x}(x)\,\frac{\mathrm{d}v}{\mathrm{d}x}(x)\,\mathrm{d}x = \int_0^1 f(x)v(x)\,\mathrm{d}x, \quad \forall v \in \mathcal{C}^1_{0,\mathrm{pw}}([0,1]), \tag{2.2.1}$$

cf. [NPDE, Eq. (1.4.19)]. For the discretization of (2.2.1) we may use a so-called *Fourier-spectral* Galerkin method, which boils down to a Galerkin method using the trial and test space

$$V_{N,0} := \operatorname{span}\{\sin(\pi x), \sin(2\pi x), \ldots, \sin(N\pi x)\}\ , \tag{2.2.2}$$

and the basis function already given in the definition (2.2.2). It is related the spectral Galerkin scheme discussed in [NPDE, Sect. 1.5.1.1].

(2.2a) Show that the functions specified in (2.2.2) really provides a basis of $V_{N,0}$.

(2.2b) Which basis should be used for the Fourier spectral scheme, if we had to approximate functions in the space $\mathcal{C}^1_{0,\mathrm{pw}}([a,b])$ for fixed $a < b$, instead of the space $\mathcal{C}^1_{0,\mathrm{pw}}([0,1])$.

HINT: Read [NPDE, Rem. 1.5.48].

(2.2c) Since (2.2.1) 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 (2.2.2). Discuss structural properties of the matrix like sparsity, symmetry, regularity.

(2.2d) Implement an efficient MATLAB function

```
function y = evaltrigsum(mu,M)
```

that evaluates the function

$$\phi(x) = \sum_{j=1}^{N} \mu_j \sin(\pi j x)$$

at the $M-1$ equidistant points $x = \frac{k}{M}$, $k = 1, \ldots, M-1$ for $M > N$.

HINT: Recall $\sin(\xi) = \frac{1}{2i}(\mathrm{e}^{\mathrm{i}\xi} - \mathrm{e}^{-\mathrm{i}\xi})$. Thus, reduce the task to a discrete Fourier transform, see [NCSE, Sect. 8.2], and use `ifft` to perform the evaluation. Its MATLAB help page will also give you valuable information. Zero-padding will be required.

(2.2e) Write a MATLAB function

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

that computes the Galerkin matrix for the Fourier spectral Galerkin scheme for (2.2.1). 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.

(2.2f) 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.

(2.2g) For $\sigma(x) = \frac{1}{\cosh(\sin(\pi x))}$, $f(x) = \pi^2 \sin(\pi x)$, determine an approximate solution of (2.2.1) 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 [NPDE, Remark 1.6.38]. 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. 1.6.30].

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

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

(2.2h) Carry out the investigations requested in subproblem (2.2g) 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 (2.2g) and try to explain.

HINT: The exact solution is

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

Listing 2.3: Testcalls for Problem 2.2

```
1 mu = [1,-1,1/2];
2 M = 6;
3
4 fprintf('\n\n##evaltrigsum:')
```

```
evaltrigsum(mu,M)

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 2.4: Output for Testcalls for Problem 2.2

```
>> test_call

##evaltrigsum:
ans =

    0.1340
         0
    0.5000
    1.7321
    1.8660

##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 2.3 $L^2(0,1)$-Orthogonal Projection onto Polynomial Space (Core problem)

This problem deals with spectral polynomial Galerkin discretization as introduced in [NPDE, Sect. 1.5.1.1] for a very simple variational problem on an interval. A moderate amount of MATLAB implementation is requested along with a numerical study of convergence.

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

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

In this problem, we will use the subspace $V_N = P_p(\mathbb{R})\big|_{[0,1]}$ of polynomials of degree p on $[0,1]$ in the context of a spectral polynomial Galerkin discretization of (2.3.1). This variant of Galerkin discretization was treated in [NPDE, Sect. 1.5.1.1] and another perusal of this part of the lecture material is recommended. As basis we use the Legendre polynomials transformed to $[0,1]$ (see [NPDE, Def. 1.5.40] and [NPDE, Code. 1.5.46]).

(2.3a) Write a MATLAB function

```
A = galmatrix_leg(N)
```

to compute the Galerkin matrix `A` with `N` degrees of freedom.

HINT: Exploit the *orthogonality relation* [NPDE, Eq. (1.5.42)]. Remember to transform the integrals to $[0,1]$ (divide by 2).

(2.3b) What is special about the Galerkin matrices computed in the previous sub-problem? Are they sparse matrices?

(2.3c) Write a MATLAB function

```
L = rhs_leg(N,f)
```

to compute the right-hand side column vector `L` with `N` degrees of freedom, where `f` is a function handle to f. As in [NPDE, Code. 1.5.64] use Gaussian quadrature with $p+1$ points (weights and nodes provided by the `gauleg` function available from the course webpage).

HINT: Use the recursion implemented in the MATLAB function `intlegpol`, see [NPDE, Code. 1.5.46], to evaluate the Legendre polynomials at given (quadrature) points. Again, remember to transform the integrals to $[0,1]$.

(2.3d) Write a MATLAB function

```
U = l2proj_leg(N,f,x)
```

that solves (2.3.1) approximately based on polynomial spectral Galerkin with a Legendre polynomial basis. The function should return the solution evaluated at the points in the vector `x` in the vector `U`.

HINT: Use your variant of `intlegpol` to evaluate the solution, as is done in `lin2pbvpspecgal.m`, see [NPDE, Code. 1.5.64].

(2.3e) Investigate the convergence for the L^∞- and L^2-norms for $f(x) = \sqrt{x}$ [NPDE, Remark 1.6.38]. Use the sampling and quadrature strategies from subproblems (2.2g) and (2.2h). Use $N = 3, \ldots, 7$.

HINT: The exact solution is $u(x) = \sqrt{x}$.
For $N = 3$, the L^∞-error should be around 0.202 and the L^2-error should be around 0.013.

Listing 2.5: Testcalls for Problem 2.3

```
N = 4;

fprintf('\n\n##galmatrix_leg:')
galmatrix_leg(N)

f = @(x)(sqrt(x));
fprintf('\n\n##rhs_leg:')
rhs_leg(N,f)

x=linspace(0,1,10)';
fprintf('\n\n##l2proj_leg:')
l2proj_leg(N,f,x)
```

Listing 2.6: Output for Testcalls for Problem 2.3

```
>> test_call_spec

##galmatrix_leg:
ans =

    1.0000         0         0         0
         0    0.3333         0         0
         0         0    0.2000         0
         0         0         0    0.1429

##rhs_leg:
ans =

    0.6678
    0.1321
   -0.0176
    0.0044

##l2proj_leg:
ans =

  Columns 1 through 7

    0.1530    0.3236    0.4634    0.5776    0.6711    0.7491
       0.8166

  Columns 8 through 10

    0.8787    0.9405    1.0071
```

Problem 2.4 Linear Finite Elements for the Brachistochrone Problem

The brachistochrone problem amounts to the minimization problem on a space of curves with fixed endpoints

$$\mathbf{u}_* = \operatorname*{argmin}_{\mathbf{u}\in V} \int_0^1 \frac{\|\mathbf{u}'(\xi)\|}{\sqrt{-u_2(\xi)}} \mathrm{d}\xi \;, \tag{2.4.1}$$

where

$$V := \{\mathbf{v} \in C^1_{\mathrm{pw}} : \mathbf{v}(0) = 0,\ \mathbf{v}(1) \text{ fixed},\ (\mathbf{v}(1))_2 < 0\} \;. \tag{2.4.2}$$

Structurally, this minimization problem is similar to the minimization problem for the potential energy of an elastic string under pinning conditions discussed in [NPDE, Sect. 1.2.3], see, in particular, [NPDE, Eq. (1.2.26)]. As in [NPDE, Sect. 1.3.1], we can also convert (2.4.1) into a variational problem and end up with

$$\mathbf{u} \in V : \quad \int_0^1 \left(\frac{\mathbf{u}' \cdot \mathbf{v}'}{\sqrt{-u_2}\|\mathbf{u}'\|} - \frac{v_2\|\mathbf{u}'\|}{2\sqrt{-u_2}u_2} \right) \mathrm{d}\xi = 0 \quad \forall \mathbf{v} \in V_0 \;, \tag{2.4.3}$$

where the test space of admissible perturbations is

$$V := \{\mathbf{v} \in C^1_{\mathrm{pw}} : \mathbf{v}(0) = \mathbf{v}(1) = 0\} \;, \tag{2.4.4}$$

because the curve may not be perturbed in the endpoints. The counterpart of (2.4.3) for the elastic string model is [NPDE, Eq. (1.3.12)].

In this problem we focus on the Galerkin discretization of (2.4.3) by means of linear finite elements as introduced in [NPDE, Sect. 1.5.1.2]. Throughout we use equidistant meshes $\mathcal{M} = \{]x_{j-1} := \frac{j-1}{M}, x_j := \frac{j}{M}[, j = 1, \ldots, M\}$ of $[0, 1]$ and the standard "tent function" basis $\mathfrak{B}$ of the Galerkin trial space

$$V_{N,0} = (\mathcal{S}^0_{1,0}(\mathcal{M}))^2 = \left\{ \begin{array}{l} \mathbf{v} \in (C^0([0,1]))^2 : \ \mathbf{v}_{|[x_{i-1},x_i]} \text{ linear} \;, \\ i = 1, \ldots, M,\ \mathbf{v}(0) = \mathbf{v}(1) = 0 \end{array} \right\} \;. \tag{2.4.5}$$

This is explained in detail in [NPDE, Ex. 1.5.92], see, in particular, [NPDE, Eq. (1.5.101)]. It is recommended to use the ordering of the basis functions implied by [NPDE, Eq. (1.5.101)], though you are free to use any other scheme.

(2.4a) 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 (2.4.3) can be written as

$$\mathbf{u} \in V : \quad \int_0^1 \sigma(\mathbf{u})(\xi)\mathbf{u}'(\xi) \cdot \mathbf{v}'(\xi)\, \mathrm{d}\xi = \int_0^1 \mathbf{f}(\mathbf{u})(\xi) \cdot \mathbf{v}(\xi)\, \mathrm{d}\xi \quad \forall \mathbf{v} \in V_0 \;. \tag{2.4.6}$$

The point of recasting 2.4.3 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, Ex. 1.5.69], Code [NPDE, Code 1.5.73], and [NPDE, Ex. 1.5.92], Code [NPDE, Code 1.5.109]. Please study the latter example and code again, in case you do not remember the rationale behind 2.4.6.

(2.4b) 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`.

(2.4c) Write a MATLAB function

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

where the input arguments are the same as in sub-problem (2.4b) and the output is a $2 \times K$ matrix, with K the length of `xi`, with 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.

(2.4d) Implement a MATLAB function

```
t = traveltime (mu)
```

which accepts as input the vector `mu` as in (2.4b), and returns the approximate value of the functional

$$J(\mathbf{u}_N) = \int_0^1 \frac{\|\mathbf{u}'_N(\xi)\|}{\sqrt{-(\mathbf{u}_N)_2(\xi)}} \mathrm{d}\xi \tag{2.4.7}$$

representing to 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 2.4.7, use the midpoint rule (see [NPDE, Eq. (1.5.88)]). 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`.

(2.4e) Proceeding as in [NPDE, Ex. 1.5.92], the discretization of 2.4.6 leads to a *nonlinear* system of equations of the form

$$\begin{pmatrix} \mathbf{R}(\vec{\boldsymbol{\mu}}) & 0 \\ 0 & \mathbf{R}(\vec{\boldsymbol{\mu}}) \end{pmatrix} \vec{\boldsymbol{\mu}} = \begin{pmatrix} \vec{\boldsymbol{\varphi}}_1(\vec{\boldsymbol{\mu}}) \\ \vec{\boldsymbol{\varphi}}_2(\vec{\boldsymbol{\mu}}) \end{pmatrix} , \tag{2.4.8}$$

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

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

such that, given the coefficients $\vec{\boldsymbol{\mu}} =$ `mu` in input (as in sub-problem (2.4a)), returns the matrix `R` $= \mathbf{R}(\vec{\boldsymbol{\mu}})$.
For the evaluation of the integrals, use the midpoint rule [NPDE, Eq. (1.5.88)].

HINT: Compute the matrix including also the rows and columns referring to the two basis functions for the offset function. In this way, you matrix `R` will have dimensions $(M+1) \times (M+1)$. Then, in subproblem (2.4g), 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 (2.4g), 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`.

(2.4f) Write a MATLAB function

```
phi = rhs(mu)
```

which, given the coefficients `mu` in input, returns as output the right hand side vector from (2.4.8)

$$\vec{\boldsymbol{\varphi}}(\vec{\boldsymbol{\mu}}) = \begin{pmatrix} \vec{\boldsymbol{\varphi}}_1(\vec{\boldsymbol{\mu}}) \\ \vec{\boldsymbol{\varphi}}_2(\vec{\boldsymbol{\mu}}) \end{pmatrix} \in \mathbb{R}^{2M-2} . \tag{2.4.9}$$

For integration, consider the composite trapezoidal quadrature rule [NPDE, Eq. (1.5.85)]. 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`.

(2.4g) The nonlinear system (2.4.9) 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, Ex. 1.5.92], [NPDE, Code 1.5.109]).
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 1.5.109], line 41).
For each iteration, plot the shape of the curve (see [NPDE, Code 1.5.109], lines 19-22).

Plot the travel time as defined in (2.4d) 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

$$\mathbf{u}(\xi) = \begin{pmatrix} \pi\xi - \sin(\pi\xi) \\ \cos(\pi\xi) - 1 \end{pmatrix} , \quad 0 \leq \xi \leq 1 , \tag{2.4.10}$$

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`.

(2.4h) 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 1.5.109], 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`.

Listing 2.7: Testcalls for Problem 2.4

```
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 2.8: Output for Testcalls for Problem 2.4

```
>> 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
```

Published on March 5.
To be submitted on March 11.

References

[NPDE] Lecture Slides for the course "Numerical Methods for Partial Differential Equations", SVN revision # 53231.

[NCSE] R. Hiptmair. Numerical methods for computational science and engineering. Lecture Slides, 2012. http://www.sam.math.ethz.ch/~hiptmair/tmp/NumCSE12_ext.pdf.