

## Homework Problem Sheet 1

### Problem 1.1 The Length of a Curve

This problem addresses techniques covered in [NPDE, Section 1.2.2], [NPDE, Section 1.2.3], and [NPDE, Section 1.3.1]:

- Construction of a continuous “energy functional” from a discrete model by a continuum limit
- Derivation of the variational equation satisfied by every minimizer of the “energy functional”

If you do not remember these techniques well, please study the corresponding sections of the lecture material before tackling this problem.

In [NPDE, Suppl. 1.2.5] we learned that the length of a curve described by a parametrization  $\mathbf{u} : [0, 1] \mapsto \mathbb{R}^2$  (a Cartesian coordinate system in the plane is taken for granted) is given by the expression

$$\ell(\mathbf{u}) = \int_0^1 \|\mathbf{u}'(\xi)\| \, d\xi. \quad (1.1.1)$$

In this problem, we derive this formula from a “discrete curve model,” namely a polygonal approximation consisting of a union of line segments. This is similar in spirit to the derivation of the elastic string potential energy from a mass-spring model, as carried out in [NPDE, Section 1.2.3], but simpler in terms of calculus.

In the second part of this problem we will conduct of first empirical study of *convergence*. This concerns the issue how close a discrete model with a finite dimensional configuration space is to the continuum limit model.

**(1.1a)** We interpolate the curve by a polygon  $P^{(n)}$ , which is the union of the line segments

$$P^{(n)} = \bigcup_{i=0}^n [\mathbf{u}(\xi_i^{(n)}), \mathbf{u}(\xi_{i+1}^{(n)})]$$

where  $\xi_i^{(n)} = \frac{i}{n+1}$  for  $i = 0, \dots, n+1$ . The total length of  $P^{(n)}$  is

$$\ell(P^{(n)}) = \sum_{i=0}^n \|\mathbf{u}(\xi_i^{(n)}) - \mathbf{u}(\xi_{i+1}^{(n)})\|. \quad (1.1.2)$$

Derive (1.1.1) by means of a limiting process as  $n \rightarrow \infty$  under the smoothness assumption that  $\mathbf{u} \in \mathcal{C}^2([0, 1])$ .

HINT: Use Taylor expansion as in [NPDE, Eq. (1.2.39)].

**Solution:** Using Taylor expansion, we see that

$$\begin{aligned}\mathbf{u}(\xi_i^{(n)}) &= \mathbf{u}(\xi_{i+\frac{1}{2}}^{(n)} - \frac{h}{2}) = \mathbf{u}(\xi_{i+\frac{1}{2}}^{(n)}) + \mathbf{u}'(\xi_{i+\frac{1}{2}}^{(n)})(-\frac{h}{2}) + \mathbf{u}''(\xi_{i+\frac{1}{2}}^{(n)})(-\frac{h}{2})^2 + \mathcal{O}(h^3), \\ \mathbf{u}(\xi_{i+1}^{(n)}) &= \mathbf{u}(\xi_{i+\frac{1}{2}}^{(n)} + \frac{h}{2}) = \mathbf{u}(\xi_{i+\frac{1}{2}}^{(n)}) + \mathbf{u}'(\xi_{i+\frac{1}{2}}^{(n)})(\frac{h}{2}) + \mathbf{u}''(\xi_{i+\frac{1}{2}}^{(n)})(\frac{h}{2})^2 + \mathcal{O}(h^3),\end{aligned}$$

and subsequently

$$\|\mathbf{u}(\xi_i^{(n)}) - \mathbf{u}(\xi_{i+1}^{(n)})\| = h\|\mathbf{u}'(\xi_{i+1/2}^{(n)})\| + \mathcal{O}(h^3),$$

where  $h = \frac{1}{n+1}$ . Inserting this in the sum yields

$$\ell(P^{(n)}) = h \sum_{i=0}^n \|\mathbf{u}'(\xi_{i+1/2}^{(n)})\| + \mathcal{O}(h^2).$$

As  $n \rightarrow \infty$ , the higher-order term will vanish, and the rest can be seen as a Riemann sum. Thus

$$\ell(\mathbf{u}) = \lim_{n \rightarrow \infty} \ell(P^{(n)}) = \int_0^1 \|\mathbf{u}'(\xi)\| d\xi.$$

**(1.1b)** Given  $\mathbf{a} \in \mathbb{R}^2$ ,  $\mathbf{b} \in \mathbb{R}^2$ , let  $V$  stand for the *affine space* of curves with fixed endpoints

$$V := \{\mathbf{v} \in (\mathcal{C}_{\text{pw}}^1([0, 1]))^2 \mid \mathbf{v}(0) = \mathbf{a}, \mathbf{v}(1) = \mathbf{b}\}.$$

Parallel to the developments of [NPDE, § 1.3.3], derive the variational equation satisfied by every minimizer  $\mathbf{u}_*$  of  $\ell(\mathbf{u})$  over  $V$ .

HINT: Use [NPDE, Eq. (1.3.5)] and proceed as in the derivation of [NPDE, Eq. (1.3.8)]. Do not forget to specify the trial and test spaces.

**Solution:** Using the hint we have

$$\|\mathbf{u}' + t\mathbf{v}'\| = \|\mathbf{u}'\| + t \frac{\mathbf{u}' \cdot \mathbf{v}'}{\|\mathbf{u}'\|} + \mathcal{O}(t^2),$$

which, when integrated from  $\xi = 0$  to  $\xi = 1$  gives

$$\ell(\mathbf{u} + t\mathbf{v}) = \ell(\mathbf{u}) + t \int_0^1 \frac{\mathbf{u}' \cdot \mathbf{v}'}{\|\mathbf{u}'\|} d\xi + \mathcal{O}(t^2).$$

So the directional derivative is

$$D_{\mathbf{v}}\ell(\mathbf{u}) = \int_0^1 \frac{\mathbf{u}' \cdot \mathbf{v}'}{\|\mathbf{u}'\|} d\xi.$$

The variational formulation is then to find  $\mathbf{u} \in V$  so that  $D_{\mathbf{v}}\ell(\mathbf{u}) = 0$  for all  $\mathbf{v}$  in the test space

$$V_0 := \{\mathbf{v} \in \mathcal{C}_{\text{pw}}^1([0, 1])^2 \mid \mathbf{v}(0) = \mathbf{v}(1) = \mathbf{0}\}.$$

**(1.1c)** Show that the function

$$\xi \mapsto (1 - \xi)\mathbf{a} + \xi\mathbf{b}, \quad 0 \leq \xi \leq 1, \quad (1.1.3)$$

satisfies the variational problem derived in sub-problem (1.1c).

**Solution:** This function has derivative  $\mathbf{u}' = \mathbf{b} - \mathbf{a}$ , so we get

$$D_{\mathbf{v}}\ell(\mathbf{u}) = \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} \cdot \int_0^1 \mathbf{v}' d\xi = \frac{\mathbf{b} - \mathbf{a}}{\|\mathbf{b} - \mathbf{a}\|} \cdot (\mathbf{v}(1) - \mathbf{v}(0)) = \mathbf{0}.$$

**(1.1d)** Write a MATLAB function

```
function length = lengthP(u,n)
```

that computes the approximate length of a curve using (1.1.2). The argument  $\mathbf{u}$  is a function handle to the curve  $\mathbf{u} : [0, 1] \mapsto \mathbb{R}^2$  and  $n$  is, as in (1.1.2), the number of internal points used for the polygonal approximation of the curve.

The function  $\mathbf{u}$  expects as input a parameter value  $\xi$  and returns the value of  $\mathbf{u}$  in that point, that is it returns  $\mathbf{u}(\xi)$ .

**Solution:** See Listing 1.1.

Listing 1.1: Code for lengthP.m

```
1 function length = lengthP(u,n)
2
3 xi = linspace(0,1,n+2);
4 for i=1:n+1
5     diff(:,i)=u(xi(i))-u(xi(i+1));
6 end
7 length = sum(sqrt(diff(1,:).^2+diff(2,:).^2));
```

**(1.1e)** Consider the endpoints  $\mathbf{a} = (0, 0)^T$  and  $\mathbf{b} = (1, 1)^T$ , and the arc connecting them:

$$\mathbf{u}(\xi) = \begin{pmatrix} 0 \\ 1 \end{pmatrix} + \begin{pmatrix} \cos\left(\xi\frac{\pi}{2} + \frac{3\pi}{2}\right) \\ \sin\left(\xi\frac{\pi}{2} + \frac{3\pi}{2}\right) \end{pmatrix}, \quad 0 \leq \xi \leq 1. \quad (1.1.4)$$

We want to test numerically in this particular case that, as we have proved in subproblem (1.1a),  $\ell(P^{(n)}) \rightarrow \ell(\mathbf{u})$  as  $n \rightarrow \infty$ , with  $\ell(P^{(n)})$  as in (1.1.2).

Write a MATLAB script `lengthcvd.m` to perform a convergence study. To do this, consider the values  $n = 2^i$ ,  $i = 2, \dots, 12$ , for the number of internal grid points, and for each of such values compute the norm  $|\ell(P^{(n)}) - \ell(\mathbf{u})|$ ; for the curve (1.1.4) the exact length is  $\ell(\mathbf{u}) = \frac{\pi}{2}$ . Make a double logarithmic (`loglog`) plot of the computed norms versus the values  $n = 2^i$ ,  $i = 2, \dots, 12$ . What do you observe? From the empirical data guess an expression in terms of  $n$  to which the error is proportional.

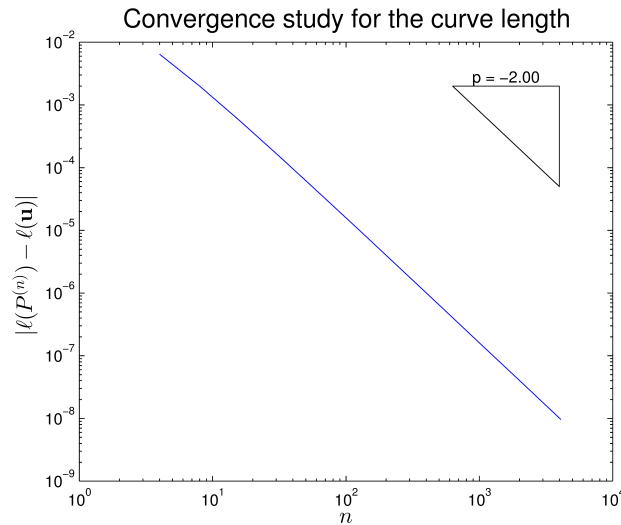


Figure 1.1: Plot for subproblem (1.1e).

**REMARK.** In [NPDE, Section 1.6] we will examine the error involved in discretization in more details and we will learn about different types of convergence of this error, see [NPDE, Section 1.6.2]. [NPDE, § 1.6.26] will discuss ways to tell the type of convergence from empiric data.

**Solution:** See Listing 1.2 for the code and 1.1 for the loglog plot. The convergence is algebraic with rate  $1.9518 \approx 2$ .

Listing 1.2: Code for lengthcvg.m

```

1 a=[0;0];
2 b=[1,1];
3 u=@(xi) [0;1]+[cos(xi*pi/2+3*pi/2); sin(xi*pi/2+3*pi/2)];
4
5 norms = [];
6 length_exact = pi/2;
7 nvals=2.^(2:12);
8
9 for n=nvals
10 lengthn = lengthP(u,n);
11 norms = [norms abs(lengthn-length_exact)];
12 end

```

**(1.1f)** If we repeat the convergence study of subproblem (1.1e) for the curve (1.1.3), than we get an error norm of the order of  $10^{-13}$  for all values of  $n$ . Try to explain such a behavior.

**Solution:** The curve is a line, so, for any value of  $n$ , its polygonal approximation coincides with the curve itself and the error  $|\ell(P^{(n)}) - \ell(u)|$  is at machine precision.

Listing 1.3: Testcalls for Problem 1.1

```

1 a=[0;0];
2 b=[1,1];

```

```

3 u=@(xi) repmat([0;1],1,length(xi))+[cos(xi*pi/2+3*pi/2);
   sin(xi*pi/2+3*pi/2)];
4
5 fprintf('\n##lengthP');
6 ln=lengthP(u,10)

```

Listing 1.4: Output for Testcalls for [Problem 1.1](#)

```

1
2 >> test_call
3
4 ##lengthP
5 ln =
6
7     1.5695

```

## Problem 1.2 Numerical Solution of Mass-Spring Model

In [\[NPDE, Section 1.2.2\]](#) we studied a mass-spring system meant to offer a discrete approximation of an elastic string, cf. [\[NPDE, Rem. 1.5.2\]](#). This problem is devoted to a numerical method for the solution of [\[NPDE, Eq. \(1.2.22\)\]](#), that is, the minimization of the total potential energy  $J^{(n)} := J_{\text{el}}^{(n)} + J_{\text{f}}^{(n)}$  with  $J_{\text{el}}^{(n)}$  from [\[NPDE, Eq. \(1.2.18\)\]](#) and  $J_{\text{f}}^{(n)}$  from [\[NPDE, Eq. \(1.2.20\)\]](#). Thus, the equilibrium positions of the point masses can be determined.

For the solution of the unconstrained minimization problem [\[NPDE, Eq. \(1.2.22\)\]](#) we rely on a so-called relaxation method. One after the other the point masses are moved to minimize the potential energy. The positions of all other masses are kept fixed. Thus we cycle through all the masses until no more substantial change of the potential energy is observed. Thus the global minimization problem is reduced to a sequence of two-dimensional minimization problems. Those are solved by Newton's method, that is, by finding a zero of the gradient of the energy functional with respect to the position of a single mass.

**(1.2a)** Write a MATLAB function

```
function plotJ(u0,u1,kappa,l,f)
```

that plots the total potential energy of a mass-string model with  $n = 1$  with respect to the position of the single free mass. The column 2-vectors  $u_0, u_1$  pass the fixed positions of the pinned ends. Both springs are equal: the numbers  $kappa, l$  provide their spring constants and equilibrium lengths, respectively. The column 2-vector  $f$  is the force acting on the free point mass.

Create a plot for the particular case when  $kappa$  and  $l$  are both equal to 1, the force  $f$  acts only in the vertical direction and has a value equal to  $-1$ , while the pins are at position  $(0, 0)$  and  $(1, 0.2)$ , see also [\[NPDE, Ex. 1.2.23\]](#).

**Solution:** See [Listing 1.5](#) for `plotJ` and [Listing 1.6](#) for how to generate the plot in [Figure 1.2](#).

Listing 1.5: Code for `plotJ.m`

```

1 function plotJ(u0, u1, kappa, l, f)
2

```

```

3      J = @(x,y) ((sqrt((x-u0(1)).^2+(y-u0(2)).^2)-l).^2 +
      (sqrt((x-u1(1)).^2+(y-u1(2)).^2)-l).^2) * (kappa/(l*2))
      - f(1)*x - f(2)*y;
4
5      xmin = min(u0(1),u1(1)) - 10;
6      xmax = max(u0(1),u1(1)) + 10;
7      ymin = min(u0(2),u1(2)) - 10;
8      ymax = max(u0(2),u1(2)) + 10;
9
10     x = linspace(xmin, xmax, 50);
11     y = linspace(ymin, ymax, 50);
12     [x,y] = meshgrid(x,y);
13     j = J(x,y);
14
15     surf(x,y,j);
16
17 end

```

Listing 1.6: Code for plotJ\_main.m

```

1 kappa = 1;
2 l = 1;
3 f = [0;-1];
4 u0 = [0 0];
5 u1 = [1 0.2];
6
7 plotJ(u0, u1, kappa, l, f)
8
9 xlabel('x')
10 ylabel('y')
11 zlabel('J(x,y)')
12 %view([-0.1 -.7 1])
13 colorbar
14 axis tight
15
16 % setup papersize / font size for nicer printint (requires
   % separate fig_prepare.m)
17 conf.myFontSize = 13;
18 conf.size = [700, 300];
19 fig_prepare(conf);
20 legend off %h = legend; h = legend(h, 'Location', 'best');
21 saveas(gcf, 'plotJ', 'eps')

```

**(1.2b)** Consider the function  $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ ,  $f(\mathbf{x}) = \frac{1}{2}(\|\mathbf{x}\| - l)^2$ . Compute the first and second derivative of this function, that is, its gradient  $\text{grad } f(\mathbf{x}) \in \mathbb{R}^2$  and its Hessian  $Hf(\mathbf{x}) \in \mathbb{R}^{2 \times 2}$ .

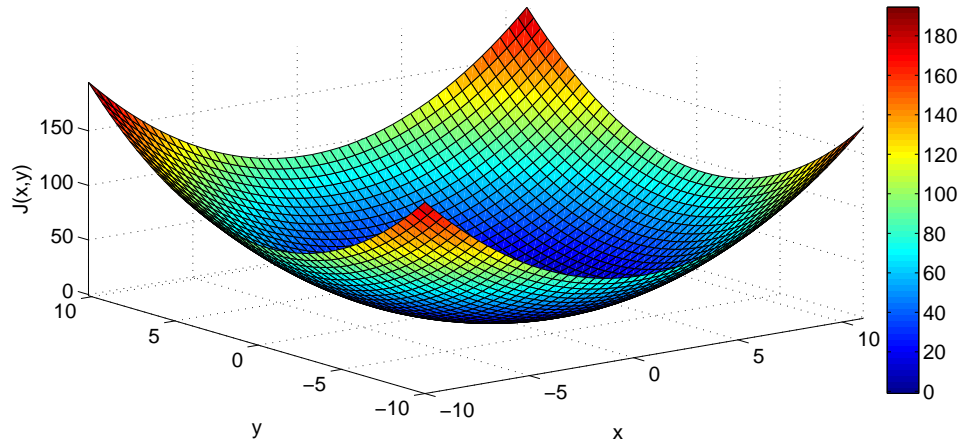


Figure 1.2: Plot of  $J(x, y)$ .

HINT: The gradient and Hessian are defined in terms of partial derivatives by

$$\mathbf{grad} f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f}{\partial x_1}(\mathbf{x}) \\ \frac{\partial f}{\partial x_2}(\mathbf{x}) \end{pmatrix}, \quad Hf(\mathbf{x}) := \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{x}) \\ \frac{\partial^2 f}{\partial x_1 \partial x_2}(\mathbf{x}) & \frac{\partial^2 f}{\partial x_2^2}(\mathbf{x}) \end{pmatrix}.$$

There are many different ways to compute these expressions:

- You may write  $f(\mathbf{x})$  explicitly as a function of  $x_1$  and  $x_2$  and obtain the partial derivatives by elementary but tedious calculus.
- You may take the cue from the multi-dimensional Taylor formula

$$f(\mathbf{x} + \mathbf{h}) = f(\mathbf{x}) + \mathbf{grad} f(\mathbf{x}) \cdot \mathbf{h} + \frac{1}{2} \mathbf{h}^T Hf(\mathbf{x}) \mathbf{h} + O(\|\mathbf{h}\|^2), \quad \mathbf{h} \in \mathbb{R}^2,$$

and rely on nested Taylor expansion of  $f(\mathbf{x})$ . You start with expanding  $\|\mathbf{x} + \mathbf{h}\|$  until a term  $O(\|\mathbf{h}\|^3)$  similar to what we did to derive [NPDE, Eq. (1.3.5)] but one step further. To do this use

$$\sqrt{1 + \tau} = 1 + \frac{1}{2}\tau - \frac{1}{8}\tau^2 + O(\tau^3).$$

**Solution:**

$$D_{\mathbf{x}} f = (\|\mathbf{x}\| - l) \frac{\mathbf{x}^\top}{\|\mathbf{x}\|} = \left(1 - \frac{l}{\|\mathbf{x}\|}\right) \mathbf{x}^\top$$

$$D_{\mathbf{x}}^\top D_{\mathbf{x}} f = \frac{l \mathbf{x} \mathbf{x}^\top}{\|\mathbf{x}\|^3} + \left(1 - \frac{l}{\|\mathbf{x}\|}\right) I$$

**(1.2c)** Compute the derivative  $D_{\mathbf{u}^i} J^{(n)}$  of  $J^{(n)}$  with respect to the position  $\mathbf{u}^i$  of the  $i$ -th mass.

HINT: This derivative is a gradient of a function  $\mathbb{R}^2 \rightarrow \mathbb{R}$ . Use the result of the previous subproblem to differentiate the elastic energy contribution.

**Solution:**

$$J^n = \frac{1}{2} \sum_{i=0}^n \frac{k_i}{l_i} (\|\mathbf{u}^{i+1} - \mathbf{u}^i\| - l_i)^2 - \sum_{i=1}^n \mathbf{f}_i \mathbf{u}_i$$

$$D_{\mathbf{u}^i} J^n = -\mathbf{f}_i^\top - \frac{k_i}{l_i} \left( 1 - \frac{l_i}{\|\mathbf{u}^{i+1} - \mathbf{u}^i\|} \right) (\mathbf{u}^{i+1} - \mathbf{u}^i)^\top + \frac{k_{i-1}}{l_{i-1}} \left( 1 - \frac{l_{i-1}}{\|\mathbf{u}^i - \mathbf{u}^{i-1}\|} \right) (\mathbf{u}^i - \mathbf{u}^{i-1})^\top$$

**(1.2d)** Compute the *second* derivative (Hessian) of  $J^{(n)}$  with respect to the position  $\mathbf{u}^i$  of the  $i$ -th mass.

HINT: Of course, the result of [subproblem \(1.2b\)](#) should be used.

**Solution:**

$$\begin{aligned} D_{\mathbf{u}^i}^\top D_{\mathbf{u}^i} J^n &= H_1 + H_2 \\ H_1 &= \frac{k_i}{l_i} \left[ \left( \frac{l_i (\mathbf{u}^{i+1} - \mathbf{u}^i) (\mathbf{u}^{i+1} - \mathbf{u}^i)^\top}{\|\mathbf{u}^{i+1} - \mathbf{u}^i\|^3} \right) + \left( 1 - \frac{l_i}{\|\mathbf{u}^{i+1} - \mathbf{u}^i\|} \right) I \right] \\ H_2 &= \frac{k_{i-1}}{l_{i-1}} \left[ \left( \frac{l_{i-1} (\mathbf{u}^i - \mathbf{u}^{i-1}) (\mathbf{u}^i - \mathbf{u}^{i-1})^\top}{\|\mathbf{u}^i - \mathbf{u}^{i-1}\|^3} \right) + \left( 1 - \frac{l_{i-1}}{\|\mathbf{u}^i - \mathbf{u}^{i-1}\|} \right) I \right] \end{aligned}$$

**(1.2e)** Verify your result for the Hessian from [subproblem \(1.2d\)](#). For a generic  $i$ ,  $i = 1, \dots, n$ , if  $\kappa_i = 1$ ,  $l_i = \frac{1}{n}$  and  $\mathbf{u}^{i+1} - \mathbf{u}^i = \mathbf{u}^i - \mathbf{u}^{i-1} = \mathbf{a} \in \mathbb{R}^2$ , you should get

$$D_{\mathbf{u}^i}^\top D_{\mathbf{u}^i} J^{(n)} := H(\mathbf{u}^i) J^{(n)} = 2 \frac{\mathbf{a} \mathbf{a}^\top}{\|\mathbf{a}\|^3} + 2 \left( n - \frac{1}{\|\mathbf{a}\|} \right) I,$$

with  $I$  the  $2 \times 2$  identity matrix.

**(1.2f)** Give the formulas for one step of the Newton method for solving (for  $\mathbf{u}^i \in \mathbb{R}^2$ )

$$\text{grad}_{\mathbf{u}^i} J^{(n)}(\mathbf{u}^1, \dots, \mathbf{u}^i, \dots, \mathbf{u}^n) = 0, \quad (1.2.1)$$

where the positions  $\mathbf{u}^1, \dots, \mathbf{u}^{i-1}, \mathbf{u}^{i+1}, \dots, \mathbf{u}^n$  are kept fixed and can be regarded as parameters.

HINT: You should still remember Newton's method from your course about elementary numerical methods. Please look it up in those course notes of yours.

**Solution:** Let  $H = H_1 + H_2$  again denote the Hessian of the potential energy function  $J^{(n)}$  and let  $H^{-1}$  be its inverse. Then, the  $m$ -th Newton's iteration for the system is:

$$\mathbf{u}_i^{m+1} = \mathbf{u}_i^m - H^{-1}(\mathbf{u}_i^m) D_{\mathbf{u}^i} J^n(\dots, \mathbf{u}_i^m, \dots)$$

**(1.2g)** Write a MATLAB function

```
function uj = locminJ(u_init, u0, u1, kappa, l, f)
```

that solves

$$\mathbf{u}^{j,*} = \underset{\mathbf{v} \in \mathbb{R}^2}{\text{argmin}} J^{(n)}(\mathbf{u}^1, \dots, \mathbf{u}^{j-1}, \mathbf{v}, \mathbf{u}^{j+1}, \dots, \mathbf{u}^n). \quad (1.2.2)$$

The position of  $\mathbf{u}^{j-1}$  and  $\mathbf{u}^{j+1}$  are passed in the  $2 \times 1$  arrays `u0` and `u1`, while a reasonable guess for  $\mathbf{u}^j$  is passed in the vector `u_init`. The row vectors `kappa` and `l` contain the spring elastic constant and equilibrium lengths, while the column vector `f` contains the force acting on  $\mathbf{u}^j$ .

HINT: Use the result of the previous sub-problem and Newton's method to find a zero of the gradient (1.2.1). Terminate the Newton iteration, if the *relative* change in the value of  $J^{(n)}$  drops below  $10^{-3}$ .

**Solution:** See [Listing 1.7](#).



Listing 1.7: Minimization problem for a single point mass

```

1  function u_r = locminJ(u_init, u0, u1, kappa, l, f)
2
3  %% Constants
4  MAXIT = 5;
5  tol = 1e-3;
6
7  %% Define the energy functionals
8  % Elastic energy
9  J_e = @(u, u0, u1, l, k) (0.5*(k(1)/l(1))*(norm(u - u0) -
10      l(1))^2 + ...
11      0.5*(k(2)/l(2))*(norm(u1 - u) - l(2))^2);
12
13  % Energy in potential field
14  J_f = @(u, f) -f'*u;
15
16  % Total energy
17  J = @(u, u0, u1, l, f, k) J_e(u, u0, u1, l, k) + J_f(u, f);
18
19  % Gradient of total energy
20  J_der = @(u, u0, u1, l, f, k) (-f - (k(2)/l(2))*(norm(u1 - u)
21      - l(2))*(u1 - u)/norm(u1 - u) + ...
22      (k(1)/l(1))*(norm(u - u0) - l(1))*(u -
23      u0)/norm(u - u0));
24
25  % Hessian of total energy
26  I2 = eye(2);
27  J_hess = @(u, u0, u1, l, f, k)
28      (k(2)/l(2)*((l(2)*(u1-u)*(u1-u)'/(norm(u1-u)^3)) + ...
29      ((1-l(2)/norm(u1-u))*I2)) + ...
30      k(1)/l(1)*((l(1)*(u-u0)*(u-u0)'/(norm(u-u0)^3))
31      + ...
32      ((1-l(1)/norm(u-u0))*I2)));
33
34  %% Use Newton to solve
35
36  u_r = u_init;
37  for i=1:MAXIT
38      % Newton correction
39      du = J_hess(u_r, u0, u1, l, f, kappa)\J_der(u_r, u0, u1, l,
40          f, kappa);
41      u_r = u_r - du;
42      if (norm(du) < tol*norm(u_r)), break; end
43  end
44
45  end

```

**(1.2h)** As already explained in the introduction to this problem, a numerical method for solving the minimization problem [NPDE, Eq. (1.2.22)] is the non-linear Gauss-Seidel relaxation. In turns the positions of the masses are adjusted to achieve a local equilibrium:

```

initial guess ( $\mathbf{u}^{1,(0)}, \dots, \mathbf{u}^{n,(0)}$ );  $k = 0$ 
do
     $k \leftarrow k + 1$ ;
    for  $j = 1 : n$ 
         $\mathbf{u}^{j,(k)} = \underset{\mathbf{v} \in \mathbb{R}^2}{\operatorname{argmin}} J^{(n)}(\mathbf{u}^{1,(k)}, \dots, \mathbf{u}^{j-1,(k)}, \mathbf{v}, \mathbf{u}^{j+1,(k-1)}, \dots, \mathbf{u}^{n,(k-1)})$ ;
    endfor
while ( $\sum_{l=1}^n \|\mathbf{u}^{l,(k)} - \mathbf{u}^{l,(k-1)}\| > \operatorname{tol} \cdot \sum_{l=1}^n \|\mathbf{u}^{l,(k)}\|$ );

```

Implement a MATLAB function

```
function u = solvemassspring(n,u0,u1,kappa,l,f,tol)
```

that computes the equilibrium shape of a mass-spring system with  $n$  free masses connected by identical springs of stiffness  $\kappa$  and equilibrium length  $l$ . This function should use the above non-linear Gauss-Seidel iteration. The meaning of the parameters  $u0, u1, \kappa, l, f$  is the same as in [subproblem \(1.2a\)](#), except that  $u1, \kappa, l$  are vectors and can potentially vary along the string. The parameter `tol` is used in the termination criterion for the non-linear Gauss-Seidel iteration.

At each iteration, make a plot of the solution.

HINT: The inner minimization step has been treated in [subproblem \(1.2f\)](#), so it can be accomplished by calling `locminJ`. As initial guess you can uniformly space your masses on the line connecting  $u0$  and  $u1$ .

**Solution:** See [Listing 1.8](#).

Listing 1.8: Solution of mass-spring model

```

1 function u = solvemassspring(n, u0, u1, k, l, f, tol)
2
3 %%%% Constants %%%%
4 MAX_ITERS = 10000;
5
6 %%%% Input validation %%%%
7 if ( any(size(u0) ~= [2,1]) || any(size(u1) ~= [2,1]) )
8     error('Wrong size for u0 or u1. The size must be [2,1]')
9 end
10
11 if (u1(1) < u0(1))
12     error('u1 must be to the right of u0 (the x-component should be larger)')
13 end

```

```

14
15 if ( any(size(l) ~= [1,n+1]) || any(size(k) ~= [1,n+1]) ||
    any(size(f) ~= [2,n]))
16     error('Wrong size for l, k or f. The sizes should be
        [1,n+1] for l and k and [2,n] for f, respectively');
17 end
18
19 %%% Init u to be distributed proportional to length %%%
20 ls = cumsum(l);
21 u_mid = [u0(1) + (u1(1) - u0(1))*ls(1:n)/sum(l); u0(2) +
    (u1(2) - u0(2))*ls(1:n)/sum(l)];
22 u_init = [u0, u_mid, u1];
23
24 %%% Energy functionals %%%
25 % J_e = @(u, l, k) elastic_energy(u, l, k);
26 % J_f = @(u, f)
    -reshape(f,1,2*n)*reshape(u(:,2:n+1),2*n,1);
27 % J = @(u, l, k, f) J_e(u, l, k) + J_f(u, f);
28
29 %%% Minimize by GAUSS-SEIDEL iterations %%%
30 u = u_init; u_old = zeros(2, n+2);
31 for i=1:MAX_ITERS
32     %plot_mass_spring_system(u);
33     plot(u(1,:),u(2,:))
34     title(sprintf('Gauss-Seidel iteration %d',i)); drawnow;
35     for j = 2:n+1
36         u(:,j) = locminJ(u(:,j), u(:,j-1), u(:,j+1), k(:,j-1:j),
            l(:,j-1:j), f(:,j-1));
37     end
38     rel_norm = norm(u - u_old, 'inf') / norm(u);
39     if rel_norm < tol
40         %sprintf('Converged !!!!!!!!!!!!!!! at relative norm =
            %g', rel_norm)
41         break;
42     end
43     u_old = u;
44     if (mod(i, 10) == 0)
45         sprintf('Relative norm = %g', rel_norm)
46     end
47 end
48
49 if (i == MAX_ITERS)
50     sprintf('Diverged !!!!!!!!!!!!!!! at relative norm =
        %g', rel_norm)
51 end
52
53 %%% Plot %%%

```

```

54 %plot_mass_spring_system(u); drawnow;
55 plot(u(1,:),u(2,:))
56 title(sprintf('mass-spring system with %d masses',n));
57
58 end

```

On the lecture Homepage you find an archive with a complete code for this problem including some very nice animations for the convergence of the Gauss-Seidel iteration, just run the main file which is called massspringdriver.

Listing 1.9: Testcalls for [Problem 1.2](#)

```

1 u0 = [0;0]; u1=[1;0.2];
2 n=10;
3
4 fprintf('\n##locminJ');
5 u_r=locminJ([u0(1); u1(2)], u0, u1, [1 1], 1/n*[1 1], [1;0])
6
7 L=1; k = ones(1,n+1);
8 f = [zeros(1,n); -(1/n)*ones(1,n)];
9 l = (L/(n+1))*ones(1,n+1);
10
11 fprintf('\n##solvemassspring');
12 u=solvemassspring(n, u0, u1, k, l, f, 10^(-1))

```

Listing 1.10: Output for Testcalls for [Problem 1.2](#)

```

1 >> test_call
2
3 ##locminJ
4 u_r =
5
6     0.5505
7     0.0976
8
9 ##solvemassspring
10 u =
11
12      0      0.0916      0.1956      0.3059      0.4177      0.5287
13      0.6383      0.7451      0.8394      0.9112      0.9617      1.0000
14      0     -0.0701     -0.1174     -0.1451     -0.1589     -0.1639
15     -0.1638     -0.1422     -0.0912     -0.0120      0.0869      0.2000

```

Published on February 18.

To be submitted on February 25.

# References

[NPDE] [Lecture Slides](#) for the course “Numerical Methods for Partial Differential Equations”.SVN revision # 73870.

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

Last modified on March 4, 2015