Winfried Auzinger
Dirk Praetorius

Sommersemester 2014
7.–9. Mai 2014

Übungsaufgaben zur VU Computermathematik

Serie 6

Einige Themen,wie z.B. plot-Befehle und einige weitere Befehle (z.B. `coeff`*,* `collect`*,* `unapply` *etc.) wurden in der VO noch nicht genauer besprochen. Dafür werden jeweils Hinweise gegeben. Konsultieren Sie für genauere Details die Maple-Hilfe!*

In den folgenden Aufgaben lässt sich das meiste durch 'Einzeiler' mittels impliziter Schleifen ausdrücken, unter Verwendung von `seq(...)`*,* `add(...)`*, etc. Sie können aber auch Steuerkonstrukte verwenden, and statt Funktionen können es auch Prozeduren (*`proc`*) sein.*

Exercise 6.1.

a) Use the command[1] `plots[listplot]` to plot the points[2] $(x_n, y_n) = \big(n, S_n - \ln(n)\big)$ in the (x, y)-plane, where the S_n are harmonic sums, $S_n = \sum_{j=1}^{n} \frac{1}{j}$.

What do you observe for larger values of n?

b) Perform an experiment to find $N \in \mathbb{N}$ such that for $n \geq N$ the relative error $(S_n - \ln n)/S_n$ is below $10\,\%$. What about $1\,\%$ accuracy?

(Motivation: Approximation of the harmonic sum S_n by a simple function.)

c) The limit $\lim_{n\to\infty} \big(S_n - \ln n\big)$ is called the *Euler-Mascheroni constant*, $\gamma \approx 0.5772156649$. Perform an experiment to find $N \in \mathbb{N}$ such that for $n \geq N$ the relative error $\big(S_n - (\ln n + \gamma)\big)/S_n$ is below $1\,‰$.

(In Maple, the constant `gamma` is predefined, and you can evaluate it using *evalf*.)

d) Determine experimentally the constant $p > 0$ such that $S_n - (\ln n + \gamma) \sim n^{-p}$ for $n \to \infty$.

Exercise 6.2.

a) Use `plots[pointplot]` to draw a regular polygon with n vertices (e.g., an equal-sided triangle for $n = 3$, a square for $n = 4$). Pack your plot command into a function `draw_polygon(n,...)` which also expects more parameters, e.g., for the color or line thickness of the plot (these are passed as options to `pointplot`). Make use of some of these parameters to produce a nice-looking plot.

b) Use `plots[pointplot3d]` to draw a cube (similar procedure as in **a)**).

Exercise 6.3. Let `A,B,C` be lists of length 2 representing Cartesian coordinates of 3 points in the plane. A (homogeneous) *barycentric representation* of the triangle $\overline{ABC}$ has the form

$$P(\alpha, \beta, \gamma) = \alpha\, A + \beta\, B + \gamma\, C, \quad \alpha + \beta + \gamma = 1,$$

with $\alpha \geq 0$, $\beta \geq 0$, and $\gamma \geq 0$.

---

[1] Syntax: `plots[listplot](...)` activates the function `listplot` contained in the package `plots`. You can also activate the complete package using `with(plots);` and simply call `listplot`.

[2] Evaluating the sum again and again is not efficient for larger values of n. In practice one would use a `for` loop for summing up.

a) Design a function `cctobc(A,B,C,x,y)` which converts Cartesian coordinates of a point (x, y) in the plane into barycentric coordinates α, β, γ with respect to A, B, C, satisfying $\alpha + \beta + \gamma = 1$. (The point (x, y) may also lie outside the triangle; in this case α, β or γ is negative.) Note that this is not well-defined if A, B, C are located on a common line.

Hint: This amounts to solving a system of two linear equations. You may derive the resulting formula by hand, or use Maple, in particular `solve`.

b) Design a function `bctocc(A,B,C,alpha,beta,gamma)` which returns the Cartesian coordinates of the point $P(\alpha, \beta, \gamma)$.

c) Choose `A,B,C` and use several calls of `pointplot` in the following way:[3]

```
p[1] := pointplot(...):  # draws the line AB
p[2] := pointplot(...):  # draws the line BC
p[3] := pointplot(...):  # draws the line CA
        ...
p[n] := pointplot(...):
```

and use `plots[display]` to render all these plots together. These plots should also include the points $P(1, 0, 0) = A$, $P(0, 1, 0) = B$, $P(0, 0, 1) = C$, and $P(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, $P(0, \frac{1}{2}, \frac{1}{2})$, $P(\frac{1}{2}, 0, \frac{1}{2})$, $P(\frac{1}{2}, \frac{1}{2}, 0)$. Produce a nice-looking plot.

Remark. The variable `gamma` is predefined and protected (see **6.1**). If you want to use the name `gamma` for one of your variables, use `unprotect(gamma)`.

Exercise 6.4. Another representation of a triangle uses non-homogeneous barycentric coordinates,

$$P = P(\lambda, \mu) = A + \lambda\,(B - A) + \mu\,(C - A), \quad \lambda + \mu \le 1,$$

with $\lambda, \mu \in [0, 1]$. The mapping $(\lambda, \mu) \mapsto P(\lambda, \mu)$ maps the unit triangle $\overline{(0, 0), (1, 0), (0, 1)}$ to $\overline{ABC}$.

a) Design two functions mapping Cartesian coordinates (x, y) into (λ, μ) and vice versa.

b) Design two functions mapping (λ, μ) coordinates into (α, β, γ) coordinates (see **6.3**) and vice versa.

c) Design a function `isintrinagle(A,B,C,x,y)` which, for a point P with Cartesian coordinates (x, y), returns `true` if P is contained in the triangle $\overline{ABC}$ and `false` otherwise.

Exercise 6.5. Let $n \in \mathbb{N}$ and consider the points $t_j := j/(n + 1)$, $j = 1 \ldots n$. Then, $\Delta := (t_1, t_2, \ldots, t_n)$ is a sequence of equispaced grid points in the interval $[0, 1]$. The functions $s_k \colon \Delta \to \mathbb{R}$, $k = 1 \ldots n$, defined by

$$s_k(t_j) = \sin(k\,\pi\,t_j), \quad j = 1 \ldots n,$$

are discrete sine functions of varying frequency.

Let $x : \Delta \to \mathbb{R}$ some function defined on Δ, and denote[4] $x_j := x(t_j)$, $j = 1 \ldots n$. The *discrete sine transform* (DST) of x is defined as the vector of 'Fourier coefficients' $\hat{x} = (\hat{x}_1, \ldots, \hat{x}_n) \in \mathbb{R}^n$,

$$\hat{x}_k = \sum_{j=1}^{n} x_j \sin(k\,\pi\,t_j), \quad k = 1 \ldots n. \tag{1}$$

[3] Each call generates a plot structure representing the data for the plot.

[4] Think of a function which takes the value 0 for $t_0 = 0$ and $t_{n+1} = 1$. The $s_k(t_j)$ also have this property.

The $\hat{x}_k$ are exactly the coefficients of a representation of x in the form

$$x = \tfrac{2}{n+1} \sum_{k=1}^{n} \hat{x}_k \, s_k \,,$$

i.e.,

$$x_j = \tfrac{2}{n+1} \sum_{k=1}^{n} \hat{x}_k \, \sin(k\,\pi\,t_j), \quad j = 1 \ldots n \,. \tag{2}$$

This means that x is decomposed into its discrete frequencies. (2) is called the *inverse discrete sine transform* (IDST); it is the same as the DST (1) up to the scaling factor $\frac{2}{n+1}$.

a) Use `pointplot` and `display` to plot the $s_k(t_j)$, $k = 1 \ldots n$, for given n, e.g., $n = 16$. (Cf. **6.3 c)**.)

b) Design two functions `[i]dst(y)` which compute the [I]DST of `y` (we represent `y` $= x$ or `y` $= \hat{x}$ by a list). 'Verify' by examples that, indeed, `idst(dst(x))=x` and vice versa. Use `pointplot` to visualize the behavior of the $\hat{x}_k$ in dependence of k.

(Natural choice for `x`: $x_j = f(t_j)$ for some given function $f(t)$ satisfying $f(0) = f(1) = 0$.)

c) 'Verify' by examples that the rescaled DST (= rescaled IDST)

$$y := \mathrm{dst}^*(x) = \Big(\sqrt{\tfrac{2}{n+1}} \sum_{j=1}^{n} x_j \, \sin(k\,\pi\,t_j), \ k = 1 \ldots n \Big)$$

is an *isometric* operation, i.e., $\sum_{k=1}^{n} y_k^2 = \sum_{k=1}^{n} x_k^2$.

Exercise 6.6. Assume you want to compute (approximate) the derivative of a given function $f(x)$ evaluated at a point $x \in \mathbb{R}$. It is assumed that f is not given explicitly as a formula but only as a ('black box') function `f`. For approximation of $f'(x)$, consider the one-sided and central difference quotients

$$\delta_h^{(1)} f(x) := \frac{f(x+h) - f(x)}{h} \,, \qquad \delta_h^{(2)} f(x) := \frac{f(x+h) - f(x-h)}{2\,h} \,,$$

with a (small) increment h.

a) Design two functions `delta1(f,x,h)` and `delta2(f,x,h)` which implement evaluation of these difference quotients. Use `evalf`.

b) Choose a function `f` with known derivative, choose a point `x` and let $h = 1, \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \ldots$. Evaluate the difference quotients and observe the speed of convergence to the true value $f'(x)$ in both cases. (Look at the error of the both approximations in dependence of h.)

c) Set `Digits:=10` (10 decimal digits floating point accuracy), and let '$h \to 0$', i.e., choose a very small increment h, $h \approx 10^{-\texttt{Digits}}$ and even smaller. What do you observe?

Exercise 6.7. The *Bernstein polynomials* of degree n are defined as

$$B_{k,n}(t) := \binom{n}{k} \, t^k \, (1-t)^{n-k} \,, \quad k = 0 \ldots n.$$

a) Design a function `B(k,n,t)` implementing the Bernstein polynomials of degree `n`.

b) Use `plot` and `display` to plot the $B_{k,n}(t)$ together for some n.

c) Design a function `Bapprox(f,n,t)` which, for a given function `f` to be approximated, returns the Bernstein-type approximation [5] of `f` of degree `n`,

$$B_n(f)(t) := \sum_{k=0}^{n} B_{k,n}(t)\, f(\tfrac{k}{n})$$

in form of a function, using the functions `B(k,n,t)` from **c)**.

d) Use `plot` and `display` to plot $f(t)$ and $B_n(f)(t)$ for some n together. Use two different colors for these curves.

Exercise 6.8.

a) Design a function `polcoe(p,z)` which expects a polynomial expression in the variable `z` as arguments and returns the list of coefficients [6] $[c_0, c_1, \ldots, c_n]$.

Hint: Use `degree` and `coeff`.

b) Design a function for the reverse operation.

c) Consider the polynomial $B_n(f)(t)$ produced in **6.7 c)** for some chosen function $f(t)$. Check what result a call of `Bapprox(f,n,t)` produces for given `n`. You can optimize this for further use in the following way: Use `collect` and `evalf` to convert the expression $B_n(f)(t)$ into a polynomial expression of the form $\sum_{k=0}^{n} c_k\, t^k$ with floating point numbers c_k. Assign this expression to a variable, say `B`, and then use the command

```
Bf := unapply(B,t);
```

(or whatever your name you choose instead of `Bf`) to convert the expression `B` into a function `Bf` of the variable t.

---

[5] It can be shown that for a function f continuous on $[0,1]$, the sequence $\{B_n(f)\}$ converges uniformly to f for $n \to \infty$, i.e., $\lim_{n\to\infty} \max_{t\in[0,1]} |B_n(f)(t) - f(t)| = 0$.

[6] Remark: In MATLAB, a polynomial $p(z) = \sum_{k=0}^{n} c_k z^k$ is represented by its coefficient vector $(c_0, c_1, \ldots, c_n)$.