Math 124 - Programming for Mathematical Applications

UC Berkeley, Spring 2023

Homework 8

Due Wednesday, March 22

In [1]:
```
using LinearAlgebra, PyPlot   # Packages needed
```

Problem 1 - Cubic splines

Consider the interpolation of $n + 1$ data points (x_i, y_i), $i = 0, \dots, n$. A *cubic spline function* $S(x)$ is a piecewise cubic polynomial, that is, if $x_j \le x \le x_{j+1}$ then $S(x) = S_j(x)$ where

$$S_j(x) = y_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

The coefficients b_j, c_j, d_j, $j = 0, \dots, n - 1$, are chosen such that the function is smooth and interpolates the given data.

Problem 1(a)

Write a function with the syntax `b,c,d = cubic_spline(x,y)` , which takes input data as vectors $\boldsymbol{x}, \boldsymbol{y}$ and solves for the coefficient vectors $\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}$ as described below. Create the matrix as a `Tridiagonal` matrix type in Julia, with the command `Tridiagonal(dl, d, du)` for lower-diagonal `dl` , diagonal `d` , and upper-diagonal `du` .

Set $h_j = x_{j+1} - x_j,\ j = 0, \ldots, n-1$, and solve the following linear system $A\boldsymbol{c} = \boldsymbol{f}$:

$$
\begin{aligned}
A &= \begin{pmatrix} 1 & 0 & & & \\ h_0 & 2(h_0 + h_1) & h_1 & & \\ & \ddots & \ddots & \ddots & \\ & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ & & & 0 & 1 \end{pmatrix} \\
\boldsymbol{f} &= (0, 3(y_2 - y_1)/h_1 - 3(y_1 - y_0)/h_0, \ldots, \\
&\qquad 3(y_n - y_{n-1})/h_{n-1} - 3(y_{n-1} - y_{n-2})/h_{n-2}, 0)^T \\
\boldsymbol{c} &= (c_0, \ldots, c_n)^T
\end{aligned}
$$

Here, $A \in \mathbb{R}^{(n+1)\times(n+1)}, f \in \mathbb{R}^{n+1}, c \in \mathbb{R}^{n+1}$, which means `d` has size `n+1` , `dl` has size `n` , and `du` has size `n` .

Finally, compute the vectors $\boldsymbol{b}, \boldsymbol{d}$ as

$$
\begin{aligned}
b_j &= (y_{j+1} - y_j)/h_j - h_j(2c_j + c_{j+1})/3 \\
d_j &= (c_{j+1} - c_j)/(3h_j)
\end{aligned}
$$

where $j = 0, \ldots, n-1$.

In [2]:

```
function cubic_spline(x,y)
    d1, dl, du, f, b, h, d = [], [], [], [], [], Float64[], []
    for i = 1:length(x) - 1
        push!(h, x[i + 1] - x[i])
    end
    d1 = [1; [h[1:end - 1][i] + h[2:end][i] for i = 1:length(h) -
    dl = [h[1:end - 1]; 0]
    du = [0; h[2:end]]

    A = Tridiagonal(dl, d1, du)
    f = [0; [3(y[i + 2] - y[i + 1])/h[2:end][i] - 3(y[i + 1] - y[i
    c = A \ f

    for i = 1:length(h)
        push!(b, (y[i + 1] - y[i])/h[i] - h[i]*(2c[i] + c[i + 1])/
        push!(d, (c[i + 1] - c[i])/(3h[i]))
    end
    return b, c, d
end
```

Out[2]: cubic_spline (generic function with 1 method)

Problem 1(b)

Write a function with the syntax `yy = spline_eval(x, y, b, c, d, xx)` which evaluates the spline defined by the data `x,y` and the computed coefficient vectors `b,c,d` at all the x-coordinates in `xx` .

In [3]:

```
function mergeLR!(L, R, x)
    # Merge the *already sorted arrays* L and R into a sorted arra
    i = j = k = 1

    # Merge L and R into x
    while i <= length(L) && j <= length(R)
        if L[i] < R[j]
            x[k] = L[i]
            i += 1
        else
            x[k] = R[j]
            j += 1
        end
        k += 1
    end

    # Copy remaining elements
    while i <= length(L)
        x[k] = L[i]
        i += 1
        k += 1
    end
    while j <= length(R)
        x[k] = R[j]
        j += 1
        k += 1
    end
end
function mergesort!(x)
    # Sort the elements of the array x using the Mergesort algorit
    if length(x) <= 1
        return x
    else
        mid = length(x) ÷ 2   # Find the midpoint of the array
        L = x[1:mid]          # Divide array into 2 halves
        R = x[mid+1:end]

        mergesort!(L)         # Sort first half
        mergesort!(R)         # Sort second half

        mergeLR!(L, R, x)
    end
    x
end
```

Out[3]:

```
mergesort! (generic function with 1 method)
```

In [4]:
```
function spline_eval(x, y, b, c, d, xx)
    xxx = mergesort!([xx; x])
    yy = []
    for i = 2:length(x)
        idx = findfirst(xxx .== x[i - 1])[1] + 1:findfirst(xxx .==
        s(v) = y[i - 1] + b[i - 1]*(v - x[i - 1]) + c[i - 1]*(v -
        append!(yy, s.(xxx[idx]))
    end
    if length(xx) != length(yy)
        i = length(x)
        lastpoints = findall(xxx .== x[end])[2:end]
        j(v) = y[i - 1] + b[i - 1]*(v - x[i - 1]) + c[i - 1]*(v -
        append!(yy, j.(xxx[lastpoints]))
    end
end

```

Out[4]: `spline_eval (generic function with 1 method)`

Problem 1(c)

Test your function by computing the spline interpolant of the function

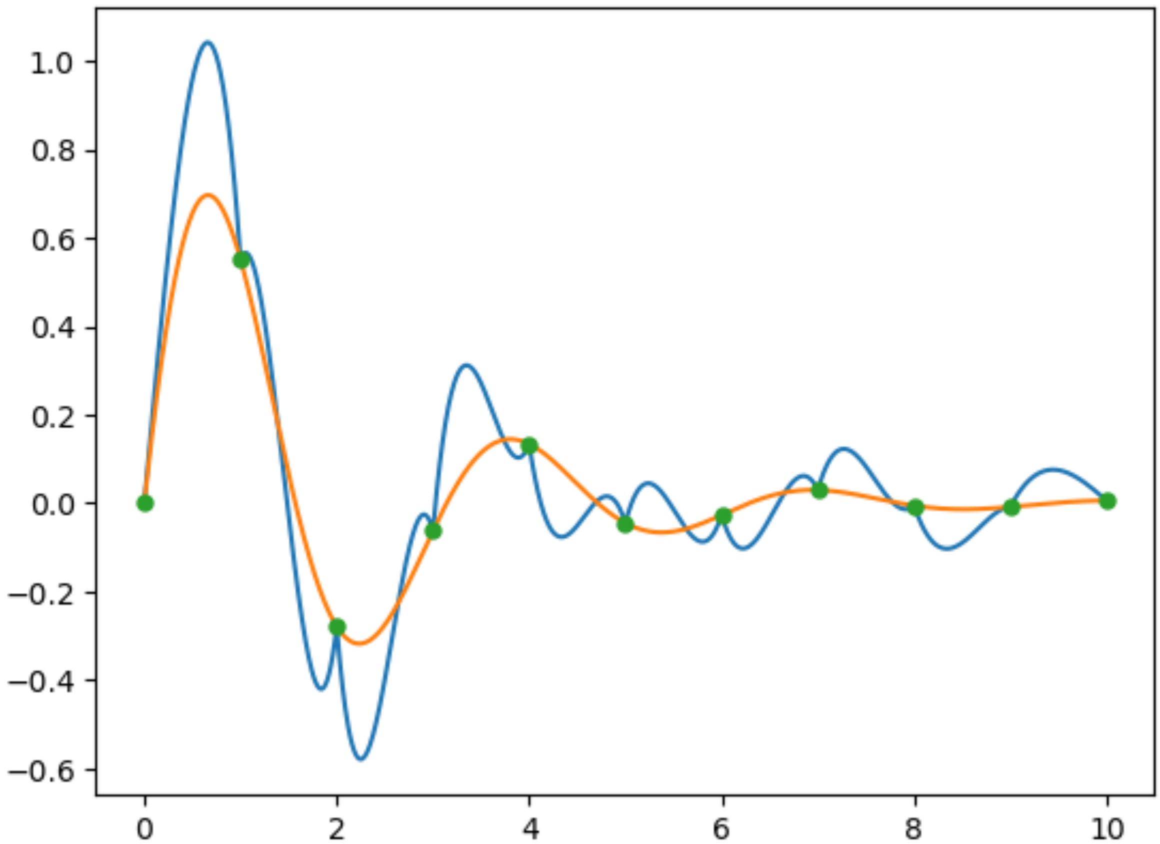
$$f(x) = e^{-x/2} \sin 2x$$

at the interpolation points $(x_i, f(x_i))$, $i = 0, \ldots, 10$.

Plot the original function and the spline interpolant on the interval $0 \leq x \leq 10$. Also plot markers at the interpolation points.

In [5]:

```
x = range(0, stop = 10, step = 1);
xx = range(0, stop = 10, step = .01);
y = [exp(-a/2)*sin(2a) for a in x]
b, c, d = cubic_spline(x,y)
yy = spline_eval(x, y, b, c, d, xx)
plot(xx, yy)
u(b) = exp(-b/2)*sin(2b)
plot(xx, u.(xx))
plot(x, u.(x), ".", markersize = 10)
```



Out[5]:

```
1-element Vector{PyCall.PyObject}:
 PyObject <matplotlib.lines.Line2D object at 0x7fdc61d39580>
```

Problem 2 - Parametric Spline Curves

To interpolate a set of data points using a *parametric spline curve*, we compute two piecewise cubic polynomials $x(t)$ and $y(t)$, where t is a parameter along the curve. For simplicity, we will let $t_i = i$ for the $n + 1$ data points (x_i, y_i), $i = 0, \ldots, n$, interpolated such that $x(t_i) = x_i$, $y(t_i) = y_i$.

Problem 2(a) - Plotting a parametric spline curve

Write a function with the syntax

```
function plot_parametric_spline(x,y; r=10)
```

which computes and plots a parametric spline for the points in the vectors `x,y`.

For the plotting, draw straight lines between the spline points $x(t), y(t)$ for $3r + 1$ equally spaced values of t between 0 and n.

Draw the splines in blue, with a line-width of 0.5. Also set `axis` to `equal`.

In [17]:

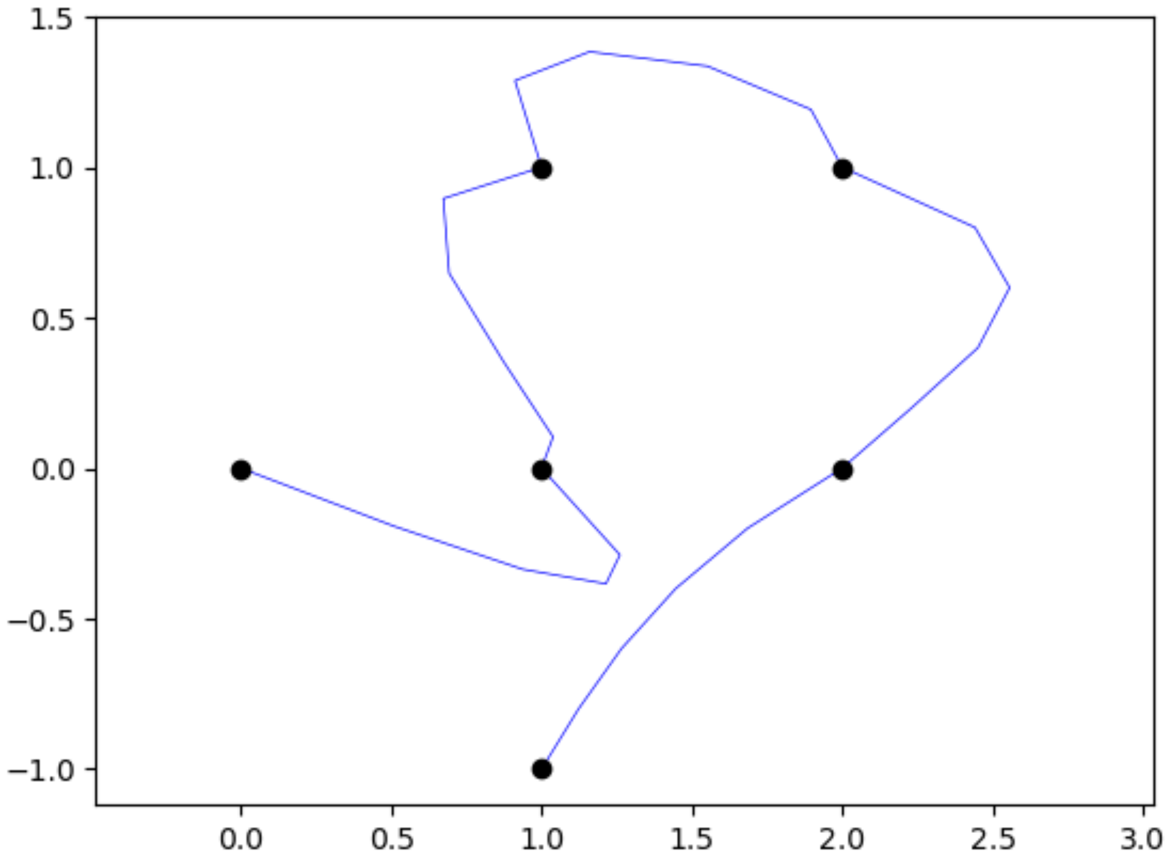
```
function plot_parametric_spline(x,y; r=10)
    t = 0:length(x) - 1
    tt = range(0, length(x), step = length(x)/(3r))
    x1, x2, x3 = cubic_spline(t,x)
    y1, y2, y3 = cubic_spline(t,y)
    xt = spline_eval(t, x, x1, x2, x3, tt)
    yt = spline_eval(t, y, y1, y2, y3, tt)

    plot(xt, yt, color = "blue", linewidth = 0.5)
    axis("equal")
end
```

Out[17]:

```
plot_parametric_spline (generic function with 1 method)
```

Test the function using the code below.

In [18]:
```
xy = [0 0; 1 0; 1 1; 2 1; 2 0; 1 -1]
plot_parametric_spline(xy[:,1], xy[:,2])
plot(xy[:,1], xy[:,2], "ko");
```



Problem 2(b) - Reading spline curves from a file

Download the file `bmw.dat` (https://raw.githubusercontent.com/popersson/math124files/main/homework/bmw.dat), and upload it to the same directory as your Julia notebook.

The file contains the coordinates for a number of splines, with each spline in the following format:

N_k
$x_1 \;\; y_1$
$x_2 \;\; y_2$
$\ldots$
$x_{N_k} \;\; y_{N_k}$

This is repeated until the file ends (which can be detected using the `eof` function). It is recommended to open the file and look at some of the lines, to make sure you know exactly how it is formatted.

Write a function

```
function read_splines(fname)
```

which opens a file `fname` containing splines as described above, and returns an array where the k-th element is an N_k-by-2 matrix with the x, y-points for each spline (note that N_k is in general different for each spline).

Hints: This is probably easiest to do using strings and the `readline` function. To convert a string `str` to an integer, use `parse(Int64, str)`. To convert two numbers in the string to a vector of two floats, use `parse.(Float64, split(str))`.

In [8]:
```
function read_splines(fname)
    lines = readlines(fname)
    v = []
    for str in lines
        if ' ' ∈ str
            push!(v, parse.(Float64, split(str)))
        else
            push!(v, parse(Int64, str))
        end
    end

    V = []

    for i in findall(typeof.(v) .== Int64)
    x = []
    y = []
        for j = i + 1:i + v[i]
            push!(x, v[j][1])
            push!(y, v[j][2])
        end
    C = [x y]
    push!(V, C)
    end
    V
end
```

Out[8]:
```
read_splines (generic function with 1 method)
```

Test your function by reading the file `bmw.dat` :

In [9]:
```
splines = read_splines("bmw.dat");
```

Problem 2(c) - Plotting an entire spline geometry

Write a function

```
function plot_splines(splines)
```

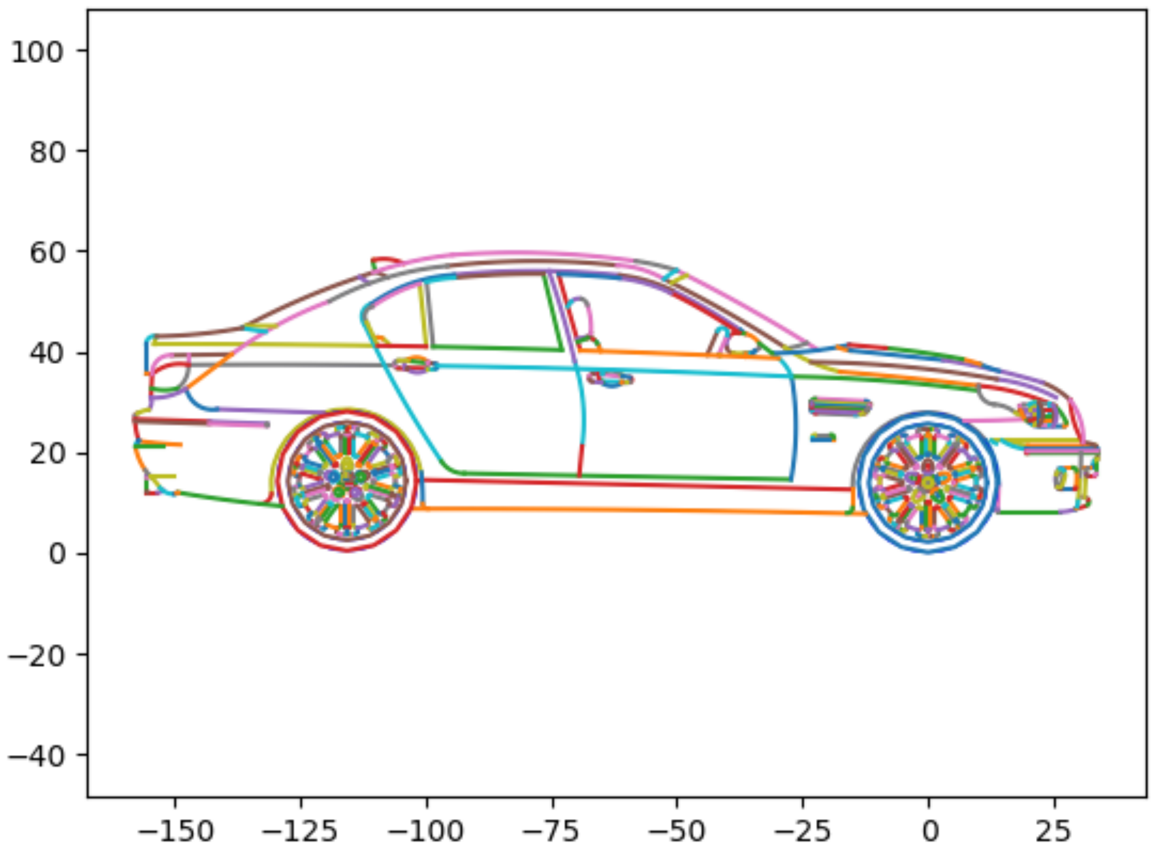
which plots all the parametric spline curves in `splines` (an array like the one returned by `read_splines`).

In [10]:
```
function plot_splines(splines)
    for xy in splines
        plot_parametric_spline(xy[:,1], xy[:,2])
        plot(xy[:,1], xy[:,2]);
    end
end
```

Out[10]:
```
plot_splines (generic function with 1 method)
```

Plot the car twice using the commands below:

In [11]:
```
plot_splines(splines)
```



In [12]:
```
plot_splines(splines);
axis([-121,-110.5,10,19]);
```