Assignment #10

Due Monday 29 November, 2021 at the start of class.

Submit on paper or by email: `elbueler@alaska.edu`

Exercise 6.1.4.

Exercise 6.1.5.

Exercise 6.2.1. Do part (b) only. Note that the "error" is $\max_i |u_i - \hat{u}(t_i)|$, as in Example 6.2.1, and *not* the local truncation error.

Exercise 6.2.2. Do parts (b) and (g) only. Same comment on "error".

Exercise 6.2.4. Do part (a) only.

Exercise 6.3.1. Do part (a) only.

Exercise 6.3.3. Do parts (a) and (b) only. A key idea is that the function you provide for the `dudt` argument needs to be a *vector*-valued function of the scalar t and the vector u. This can be done at the command line or by saving a function in a `.m` file. In either case, show me the code you used and how you ran it from the command line.

P13. **(a)** Watch the following video, *The Beauty of Bezier Curves* by Freya Holmér:

www.youtube.com/watch?v=aVwxzDHniEw

For parts below, all the important stuff occurs in the first 7 minutes, but the whole thing is 25 easy minutes! (*Nothing to turn in. Full credit for* ***(a)*** *if you make progress below.*)

(b) Suppose P_0 and P_1 are points in the plane, so $P_0 = (x_0, y_0)$ and $P_1 = (x_1, y_1)$, and think of such points as vectors. The line segment between P_0 and P_1 is the vector-valued function $F(t) = (1-t)P_0 + tP_1$. Note that $F(t)$, which she calls lerp(P_0,P_1,t), is a linear combination of P_0 and P_1. A *quadratic Bezier curve* with control points P_0,P_1,P_2 is a linear combination of F functions:

$$Q(t) = (1-t)\left[(1-t)P_0 + tP_1\right] + t\left[(1-t)P_1 + tP_2\right]$$

Expand this expression and show that it is quadratic in t. In fact, you can factor-out the points and write it as

$$Q(t) = P_0\,\beta_0(t) + P_1\,\beta_1(t) + P_2\,\beta_2(t) = \sum_{j=0}^{2} P_j\,\beta_j(t).$$

Find $\beta_0(t)$, $\beta_1(t)$, and $\beta_2(t)$.

(c) A *cubic Bezier curve* with control points P_0,P_1,P_2,P_3 is a linear combination of quadratic Bezier curves:

$$\begin{aligned} P(t) = (1-t)\left\{(1-t)\left[(1-t)P_0 + tP_1\right] + t\left[(1-t)P_1 + tP_2\right]\right\} \\ + t\left\{(1-t)\left[(1-t)P_1 + tP_2\right] + t\left[(1-t)P_2 + tP_3\right]\right\}. \end{aligned}$$

Writing it this way is what she calls "De Casteljau's algorithm." As she says, it is numerically stable because it is just "lerps all the way down." Expand $P(t)$ in order to compute $b_j(t)$ in the expression

$$P(t) = \sum_{j=0}^{3} P_j\, b_j(t).$$

You will be able to check your formulas via the video. The functions $b_j(t)$ are the cubic *Bernstein polynomials*. Note that she visualizes this representation using four colors for the weights, which are the values $b_j(t)$.

(d) Confirm that $P(0) = P_0$ and $P(1) = P_3$, that is, confirm that the cubic curve starts at control point P_0 and ends at P_3. Note that the curve *does not*, however, hit P_1 and P_2.

(e) Write you own cubic Bezier curve code in Matlab. In particular, write a function

```
function bezier(p0, p1, p2, p3)
```

where each px is a pair of numbers. That is, the input is four points. The function should simply generate a figure showing the curve, and also the four points as markers. (*There is no need to return numbers. You can use either of the forms above for $P(t)$.*) When you test it, try to approximately reproduce one of the figures from the video; mention which minute in the video is showing a similar curve.