Numerical Methods (CS 357)

Worksheet

Part 1. Objectives

- Understand contributions to error in interpolation
- Be able to asymptotically predict interpolation error
- Understand ways in which polynomial interpolation can fail or yield large errors
- Understand how piecewise polynomial interpolation and appropriate choice of nodes mitigate sources of error

Part 2. Interpolation: Error prediction

Suppose we interpolate a function $f(x)$ using monomials at three equally-spaced points on an interval of length h. The obtained interpolation error is found to be 0.5. What interpolation error do you predict for the same function on a subinterval of the original one that has length $h/10$?

Part 3. Finding a 2-piece quadratic interpolant

In this problem, you are given three data points at equispaced x-coordinates x_0, x_1, x_2 in the variable `x` with associated y values y_0, y_1, y_2 in the variable `y`.

From this, you should find a piecewise quadratic interpolant by setting up a Vandermonde matrix V and finding the coefficients a, b, c and d, e, f, so that your interpolant is

$$\tilde{f}(x) = \begin{cases} ax^2 + bx + c & x < x_1 \\ dx^2 + ex + f & x \geq x_1 \end{cases}$$

Your interpolant should obey the following conditions:

- $f'(x_0) = 0$
- $\tilde{f}(x_i) = y_i$
- $\tilde{f}'$ does not jump between pieces

INPUT:

- `x`, a 3-vector of x coordinates
- `y`, a 3-vector of function values $f(x)$

OUTPUT:

- `coeffs`, the vector $[a, b, c, d, e, f]$ of coefficients

```
import numpy as np
import numpy.linalg as la
import matplotlib.pyplot as pt

V = np.zeros((6,6))
rhs = np.zeros(6)

# Fill first row of linear system: a*x[0]**2 + b*x[0] + c = y[0]
V[0,:3] = [x[0]**2, x[0], 1]
rhs[0] = y[0]

# a*x[1]**2 + b*x[1] + c = y[1]


# d*x[1]**2 + e*x[1] + f = y[1]


# d*x[2]**2 + e*x[2] + f = y[2]


# 2*a*x[0] + b = 0


# 2*a*x[1] + b - 2*d*x[1] - e = 0


print(V)
coeffs =

# (plot the solution--no need to modify this part)
a, b, c, d, e, f = coeffs
yp = np.empty_like(xp)
left = xp < x[1]; lxp = xp[left]; yp[left] = a*lxp**2 + b*lxp + c
right = xp >= x[1]; rxp = xp[right]; yp[right] = d*rxp**2 + e*rxp + f

pt.plot(xp, yp)
pt.plot(x, y, "o")
```