Homework 01 Solution: Interpolation Theory

Assigned: 2019-02-01 17:30:00

Due: 2019-02-08 23:59:00

Points: 90 pts is full credit (10 pts extra available)

Instructions: All of this assignment will be submitted via Canvas. Handwritten solutions will receive 0 credit.

Deliverables: For this assignment, you are expected to upload a PDF file to canvas along with all figures and .tex files used to generate that pdf. Submit any codes you wrote to solve the 4x4 systems in problem 6 to Canvas.

Collaboration: This assignment is to be completed by yourself, however, you make seek assistance from your classmates. In your submission you must indicate from whom you received assistance.

Honor Code: By submitting this assignment, you acknowledge that you have adhered to the Virginia Tech Honor Code and attest to the following:

> I have neither given nor received unauthorized assistance on this assignment. The work I am presenting is ultimately my own.

Background

This assignment focuses on changing the resolution of photos.

First, we will look at subsampling, the simplest way to reduce the number of pixels in each direction by an integer factor by simply keeping only a subset of pixels. For instance, subsampling by a factor of 2 would mean keeping every other pixel in each direction.

Then we will look at interpolation, which allows you more flexibility to move your photo onto a different grid which is not necessarily a subset of the existing grid, and which may even have more pixels than the original grid. We will study three kinds of interpolation:

- Piece-wise constant
- Piece-wise linear
- Piece-wise cubic polynomial

References

- Writing pseudo-code in Latex: https://en.wikibooks.org/wiki/LaTeX/Algorithms
- Making tables in Latex: https://en.wikibooks.org/wiki/LaTeX/Tables
- Basics of 1D interpolation: https://ocw.mit.edu/courses/mechanical-engineering/2-086-numerical-computatio nutshells-guis/MIT2_086F14_Interpolation.pdf
- Basics of 2D linear interpolation: http://www.aip.de/groups/soe/local/numres/bookcpdf/c3-6. pdf

Problems

1. **Subsampling in 1D**: Consider $f : \mathbb{R} \to \mathbb{R}$. Say that you know the value f at N points $\{(x_i, f(x_i))\}_{i=0}^{N-1}$. You wish to downsample this set of points by a factor of K, keeping every K^{th} sample starting with $i = C$ (where $C \leq N - 1$).

   (a) (5 points) What are the indices of the set of points that are kept after subsampling?

   Answer: Starting at C and stopping at or before $N - 1$ gives the set of M indices

   $$\{i | i = C + jK \text{ such that } j = 0, 1, 2, \ldots, M - 1\}.$$

   (b) (5 points) Let M be the number of points that are kept after subsampling. Write a formula for M in terms of N, C and K. *Hint:* You may find it useful to use the floor $\lfloor \cdot \rfloor$ or ceiling $\lceil \cdot \rceil$ functions.

   Answer:

   $$M = \lceil \tfrac{N-C}{K} \rceil$$

   For a sanity check, consider the case of $N = 3$ and the (C, K) pairs $(0, 1)$, $(0, 2)$, $(1, 2)$, and $(2, 2)$.

   (c) (5 points) Let `fArr` be an array storing $\{f(x_i)\}_{i=0}^{N-1}$. Write pseudo-code for a function that takes `fArr`, N, C and K as inputs and outputs `fSubArr` that's the subsampled result.

   Answer:

   ```
   function DOWNSAMPLE_1D(fArr, N, C, K)
       M ← CEIL((N − C)/K)

       for î ← 0, ..., M − 1 do
           i ← C + îK
           fSubArr[î] ← fArr[i]
       end for
       return fSubArr
   end function
   ```

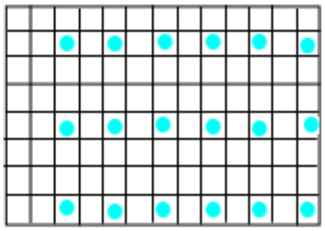


Figure 1: This grid represents the original 2D array, with blue dots in the entries that are kept by subsampling. The parameters for this subsampling are $C = 2$, $D = 1$, $K_x = 2$, $K_y = 3$, $N_x = 13$ and $N_y = 8$, with $(i, j) = (0, 0)$ in the top left corner. Note: x values increase going to the right and y values increase going downt.

2. **Subsampling in 2D**: Consider $g : \mathbb{R}^2 \to \mathbb{R}$. Note that $g(x, y)$ on an interval $[a, b] \times [c, d]$ could represent the value of a grayscale photo at position $(x, y) \in [a, b] \times [c, d]$. Say that you know the value g at $N_x N_y$ points $(x_i, y_j, g(x_i, y_j))$ for $i = 0, 1, \ldots, N_x - 1$ and $j = 0, 1, \ldots, N_y - 1$. You wish to downsample this set of points by a factor of K_x in the x direction and K_y in the y direction, starting with $i = C$ and $j = D$. As an example, see Figure 1.

   (a) (5 points) Let $N_x N_y$ be the number of points in the original set. Write a formula for $M_x M_y$, the number of points in the subsampled image, in terms of N_x, N_y, K_x, K_y, C and D.

   Answer:

$$M_x = \lceil \tfrac{N_x - C}{K_x} \rceil$$
$$M_y = \lceil \tfrac{N_y - D}{K_y} \rceil$$
$$M_x M_y = \lceil \tfrac{N_x - C}{K_x} \rceil \lceil \tfrac{N_y - D}{K_y} \rceil$$

   (b) (5 points) Let `gArr` be a 2D array storing $g(x_i, y_i)$ for $i = 0, 1, \ldots, N_x - 1$ and $j = 0, 1, \ldots, N_y - 1$. Write pseudo-code for a function that takes `gArr`, N_x, N_y, C, D, K_x and K_y as inputs, and outputs `gSubArr`, a 2D array that's the subsampled result.

   Answer: Note, we are assuming that arrays are accessed with two indices and the x-index comes first: `gArr[i,j]`.

```
1:  function DOWNSAMPLE_2D(gArr, N_x, N_y, C, D, K_x, K_y)
2:      M_x ← CEIL((N_x − C)/K_x)
3:      M_y ← CEIL((N_y − C)/K_y)
4:
5:      for î ← 0, ..., M_x − 1 do
6:          i ← C + î K_x
7:          for ĵ ← 0, ..., M_y − 1 do
8:              j ← C + ĵ K_y
9:          end for
10:         gSubArr[î, ĵ] ← gArr[i, j]
11:     end for
12:     return gSubArr
13: end function
```

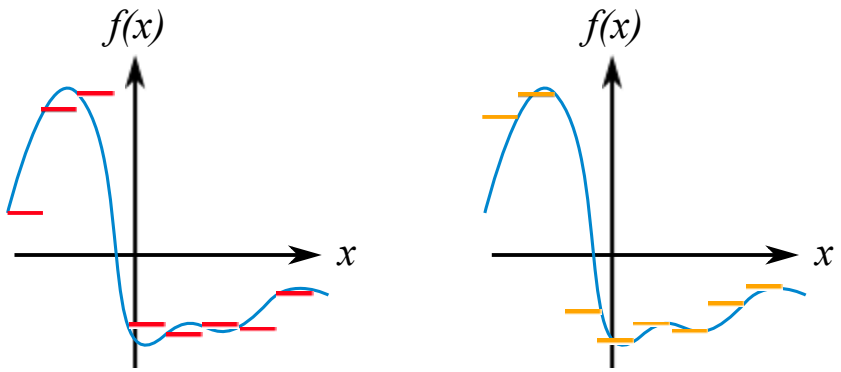


Figure 2: (Left) Left-endpoint piece-wise constant interpolation approximates f at each point by the nearest sampled point to the left side. (Right) Right-endpoint piece-wise constant interpolation approximates f at each point by the nearest sampled point to the right side.

3. **Piece-wise constant interpolation in 1D**: Consider a smooth 1D function $f : \mathbb{R} \to \mathbb{R}$. Say that we don't know the continuous form of f, and only have the sampled values of f at a set of N evenly-spaced points $\{(x_i, f(x_i))\}_{i=0}^{N-1}$ where $x_i = a + hi$ and $h = \frac{b-a}{N-1}$. If one wishes to estimate $f(\hat{x})$ by $\hat{f}(\hat{x})$ where $x_i \leq \hat{x} \leq x_{i+1}$, it would be simplest to set $\hat{f}(\hat{x})$ equal to one of the neighboring values. We would need to choose a convention, do we set it equal to $f(x_i)$ or $f(x_{i+1})$? These two options are called *left-endpoint* and *right-endpoint piece-wise constant interpolation*. A true f versus its left-endpoint and right-endpoint approximations are shown in Figure 2. *Note*: You'll notice that if your new grid is a subset of your old grid, constant interpolation yields the same results as subsampling (i.e. subsampling is a special case of constant interpolation).

   (a) (10 points) Let $[a, b] = [0, 3\pi]$, and $f(x) = \sin(x)$. You know the true value of f at the points $x_i = 0 + hi$ including $x_0 = 0$ and $x_N = 3\pi$.

   - Let $h = \frac{2\pi}{3}$. Using right-endpoint piece-wise constant interpolation, which x_i would you use to estimate the value of f at $x = 3$? Estimate the value of f at $x = 3$. What is the error from the true value (note: you may use a calculator)?

     Answer: $h = 2\pi/3$ implies that $\{x_0, x_1, x_2, x_3, x_4\} = \{0, 2\pi/3, 4\pi/3, 2\pi, 8\pi/3\}$. [a] Of these points, $\hat{x} = 3$ is right-closest to $4\pi/3$, so the error is

     $$|\sin(4\pi/3) - \sin(3)| \approx 1.007145.$$

     [a] Note that 3π is not a point where we have data here, but that is OK. While nearest-right neighbor interpolation is not defined for the end of the interval, nearest-left neighbor is defined here.

   - Let the error at x be defined by $e(x) = |\hat{f}(x) - f(x)|$. What is $\max_{[0,3\pi]} e(x)$ when $\hat{f}$ is left-endpoint piece-wise constant interpolation for $h = \frac{2\pi}{3}$, and at what x value(s) does the largest error occur? When $h = \frac{\pi}{3}$? When $h = \frac{\pi}{6}$? Display your results in a table with three columns and a row at the top with column headers.

     Answer: First, we consider the case where $h = 2\pi/3$. $\hat{f}(x)$ is a piecewise constant function. Therefore, using the provided definition gives

     $$e(x) = \begin{cases} |\sin(x_0) - \sin(x)| & x \in [x_0, x_1) \\ |\sin(x_1) - \sin(x)| & x \in [x_1, x_2) \\ |\sin(x_2) - \sin(x)| & x \in [x_2, x_3) \\ |\sin(x_3) - \sin(x)| & x \in [x_3, x_4) \\ |\sin(x_4) - \sin(x)| & x \in [x_4, 3\pi] \end{cases}$$

From calculus, we know that the extrema of the function occur where $\frac{d}{dx}e(x) = 0$ or at the endpoints. e is not differentiable, but we can still look for the maxima in each interval. The maximum error will be the maximum of the maximum errors for each interval. We must be careful, however, because error is also discontinuous in the middle of the interval $[4\pi/3, 2\pi)$, at $x = 5\pi/3$. We can thus define a piecewise differentiable error function

$$e(x) = \begin{cases} \sin(x) - \sin(0) & x \in [0, 2\pi/3) \\ \sin(2\pi/3) - \sin(x) & x \in [2\pi/3, 4\pi/3) \\ \sin(4\pi/3) - \sin(x) & x \in [4\pi/3, 5\pi/3) \\ \sin(x) - \sin(4\pi/3) & x \in [5\pi/3, 2\pi) \\ \sin(x) - \sin(2\pi) & x \in [2\pi, 8\pi/3) \\ \sin(8\pi/3) - \sin(x) & x \in [8\pi/3, 3\pi] \end{cases}.$$

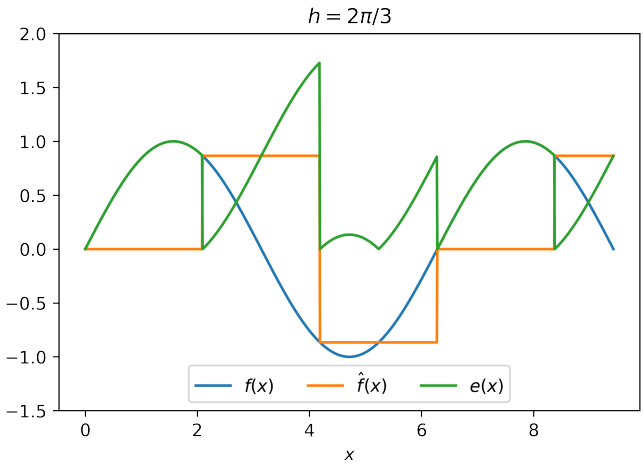
Thus, the derivative is

$$\frac{d}{dx}e(x) = \begin{cases} \cos(x) & x \in [0, 2\pi/3) \\ -\cos(x) & x \in [2\pi/3, 4\pi/3) \\ -\cos(x) & x \in [4\pi/3, 5\pi/3) \\ \cos(x) & x \in [5\pi/3, 2\pi) \\ \cos(x) & x \in [2\pi, 8\pi/3) \\ -\cos(x) & x \in [8\pi/3, 3\pi] \end{cases}.$$

Considering each interval:

- $[0, 2\pi/3)$: $\cos$ has a zero at $\pi/2$, which is a maximum of $\sin$ and therefore the error function. The end points of the interval must be smaller, therefore the maximum error on this interval is $e_{\text{max}} = 1.0$.
- $[2\pi/3, 4\pi/3)$: There are no zeros of $\cos$, so the maximum error must be on the endpoints. The error on the left endpoint is zero (by definition of the nearest left-neighbor interpolation). The right point is not in the interval, so we consider the limit of the error as we approach that point, which is $e_{\text{max}} = \sin(2\pi/3) - \sin(4\pi/3) \approx 1.73205$.
- $[4\pi/3, 5\pi/3)$: On this interval, there is a zero of $\cos$ at $x = 3\pi/2$, so the maximum error is $e_{\text{max}} = \sin(4\pi/3) - \sin(3\pi/2) \approx 0.13397$.
- $[5\pi/3, 2\pi)$: Again, the maximum error occurs as we approach $x = 2\pi$ from the left and $e_{\text{max}} = \sin(2\pi) - \sin(4\pi/3) \approx 0.866025$.
- $[2\pi, 8\pi/3)$: This interval behaves the same as the first interval, so $e_{\text{max}} = 1.0$.
- $[8\pi/3, 3\pi]$: This interval is like the second, except it includes the right point, so the max error is at $x = 3\pi$ and $e_{\text{max}} = \sin(8\pi/3) - \sin(3\pi) \approx 0.866025$.

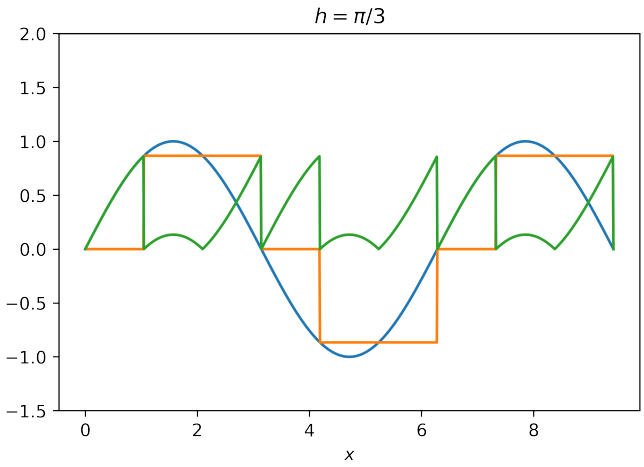
Therefore, the maximum error occurs in the limit as we approach $x = 4\pi/3$ from the left. We can see this by looking at the following plot of $f(x)$, $\hat{f}(x)$, and $e(x)$.

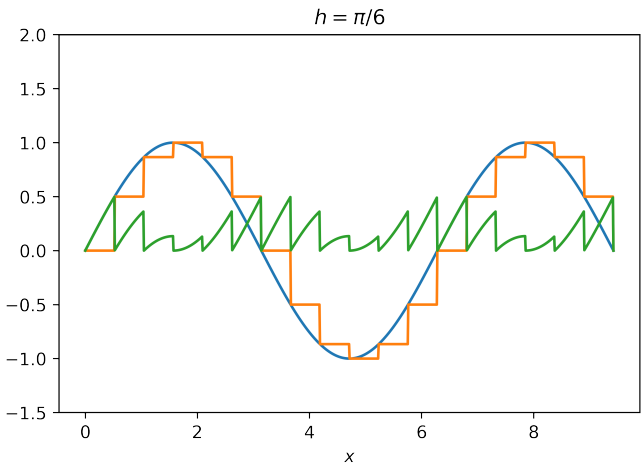


Following similar analysis as the first case, we obtain the following table, where the maxima occur as we approach the following $x-$values from the left:

	$h = 2\pi/3$	$h = \pi/3$	$h = \pi/6$
x	$4\pi/3$	$\pi/3,\ \pi,\ 4\pi/3,\ 2\pi,\ 5\pi/3,\ 3\pi$	$\pi/6,\ \pi,\ 7\pi/6,\ 2\pi,\ 13\pi/6,\ 3\pi$

These results are easily observed by the following plots:





(b) (10 points) Let `fArr` be an array storing $\{f(x_i)\}_{i=0}^{N-1}$ for $x_i = a + hi$ with $i = 0, 1, \ldots, N-1$. Now, you wish to use left-endpoint piece-wise constant interpolation to estimate f at a new set of points $\hat{x}_i = a + \hat{h}i$ with $i = 0, 1, \ldots, M-1$ and $\hat{h} = \frac{b-a}{M-1}$. Write pseudo-code for a function that takes `fArr`, N, M, a, and b as inputs and outputs `fHatArr` that's the interpolated array with M entries representing the estimated value of f at each $\hat{x}_i$.

Answer:

```
1: function INTERP_NEAREST_LEFT_1D(a, b, fArr, N, M)
2:     h ← (b − a)/(N − 1)
3:     ĥ ← (b − a)/(M − 1)
4:     for î ← 0, ..., M − 1 do
5:         x_î ← a + îĥ
6:         i ← FLOOR((x_î − a)/h)
7:         fHatArr[ î ] ← fArr[ i ]
8:     end for
9:     return fHatArr
10: end function
```

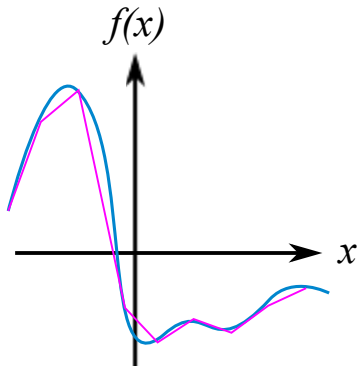


Figure 3: Linear approximation (magenta) is closer to the true function (blue) than constant interpolation.

4. **Linear interpolation**: Using the same setup as problem 3, if one wishes to estimate $f(\hat{x})$ by $\hat{f}(\hat{x})$ where $x_i \leq \hat{x} \leq x_{i+1}$, a slightly more accurate approximation would be to use a linear combination of $f(x_i)$ and $f(x_{i+1})$ (instead of just information from a single neighbor). Assuming that we want a linear model of the form $\hat{f}(\hat{x}) = \alpha\hat{x} + \beta$ that is exactly equal to the function values at x_i and x_{i+1}, we can set up and solve a two-by-two system for α and β:

$$\begin{aligned} f(x_i) &= \alpha x_i + \beta \\ f(x_{i+1}) &= \alpha x_{i+1} + \beta \end{aligned}$$

This leads to the formula $\hat{f}(\hat{x}) = \omega f(x_i) + (1-\omega) f(x_{i+1})$ where $\omega = (x_{i+1} - \hat{x})/(x_{i+1} - x_i)$. Approximating f on a new grid of points with this formula is called *piece-wise linear interpolation*. A true f versus the values of its piece-wise linear interpolation function is shown in Figure 3, and we see that the linear interpolation function is much closer than either of the piece-wise constant interpolation functions on the same grid.

(a) (10 points) Again consider $[a, b] = [0, 3\pi]$, and $f(x) = \sin(x)$. You know the true value of f at the points $x_i = 0 + hi$ including $x_0 = 0$ and $x_N = 3\pi$.

- Let $h = \frac{2\pi}{3}$. Using linear interpolation, estimate the value of f at $x = 3$. What is the error from the true value? How does this compare to the estimation error of constant interpolation in 3(a)?

  Answer: This problem uses the same set of points as in Problem 3(a). For these points, we have that $2\pi/3 = x_1 < x < x_2 = 4\pi/3$, so $\omega = (4\pi/3 - 3)/(2\pi/3)$. Then, using the formula for linear interpolation, we find the error

  $$|\omega \sin(2\pi/3) + (1-\omega)\sin(4\pi/3) - \sin(3)| \approx 0.0240238.$$

- Let the error at x be defined by $e(x) = |\hat{f}(x) - f(x)|$. What is $\max_{[0,3\pi]} e(x)$ when $\hat{f}$ is left-endpoint piece-wise constant interpolation for $h = \frac{2\pi}{3}$, and at what x value(s) does the largest error occur? When $h = \frac{\pi}{3}$? Display your results in a table with three columns and a row at the top with column headers. How do these compare to the max errors from constant interpolation in 3(a)?

  Answer: Following similar analysis as for 3(a), we seek the extrema of the error function on each interval. However, for piece-wise linear interpolation, we know that we will fit $f(x)$ exactly on the end-points of each interval, so we must find the locations in each interval where a maximum is attained.
  On the interval from x_i to x_{i+1}, the error function is

  $$e(x) = |\frac{x_{i+1} - x}{h}\sin(x_i) + (1 - \frac{x_{i+1} - x}{h})\sin(x_{i+1}) - \sin(x)|.$$

Away from the zeros, assuming the approximation is larger than the true function,

$$\frac{d}{dx}e(x) = \frac{-1}{h}\sin(x_i) + (1+\frac{1}{h})\sin(x_{i+1}) - \cos(x).$$

The extrema are at the zeros of this function, which yields the location

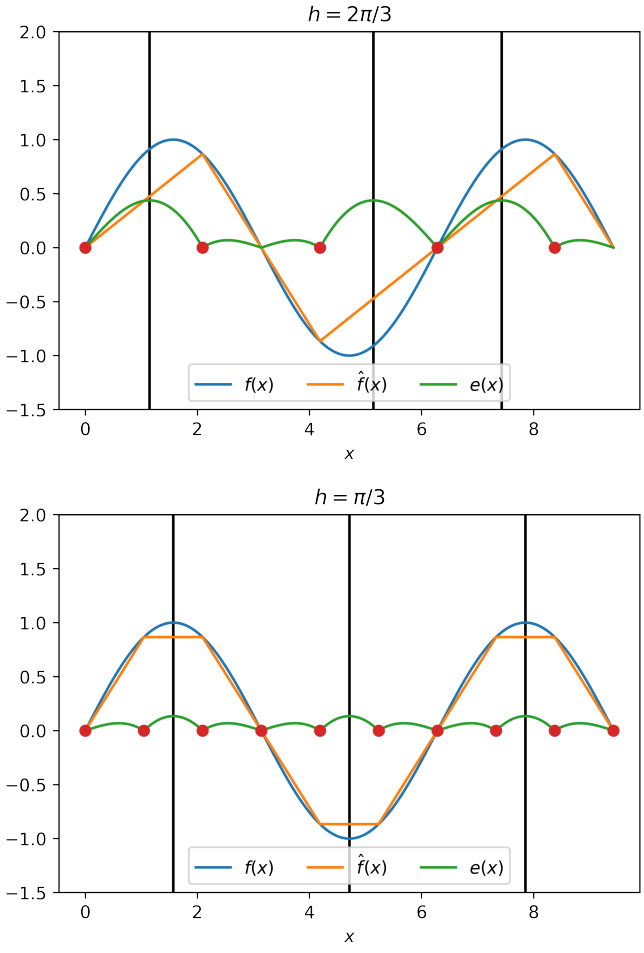
$$\hat{x} = \arccos(-\frac{1}{h}\sin(x_i) + \frac{1}{h}\sin(x_{i+1})).$$

Consider three facts: 1) $\arccos$ has a range of $[0, \pi]$, 2) $\sin$ and the interpolant are both symmetric about π for this problem, and 3) $\sin$ and the interpolant, for the values of h specified, are periodic. Hence, if the maximum error is achieved at $\hat{x}$ as above, it is also achieved at $2\pi - \hat{x}$, $2\pi + \hat{x}$, and so on.

Applying this method to each interval for $h = 2\pi/3$ and $h = \pi/3$ gives the following locations:

	$h = 2\pi/3$	$h = \pi/3$
$\hat{x}$	≈ 1.144505	$\pi/2$
x	$\hat{x}$, $2\pi - \hat{x}$, $2\pi + \hat{x}$	$\hat{x}$, $2\pi - \hat{x}$, $2\pi + \hat{x}$

This is easily visualized by the following plots:



The error of linear interpolation is more than 40 times smaller than that of the piecewise constant interpolation.

(b) (10 points) Consider a function $g : \mathbb{R}^2 \to \mathbb{R}$. You have sampled the regularly spaced points $(x_i, y_j, g(x_i, y_j))$ for $i = 0, 1, \ldots, N_x - 1$ and $j = 0, 1, \ldots, N_y - 1$ with $x_0 = a$, $x_{N_x-1} = b$, $y_0 = c$, and $y_{N_y-1} = d$. Let `gArr` be an $N_x \times N_y$ 2D array storing $g(x_i, y_j)$. You want to use linear interpolation to represent g on a new grid of $M_x \times M_y$ points spanning the same region $[a, b] \times [c, d]$. A simple way to do this is to first do 1D linear interpolation in the x-direction for each y_j value (i.e. (x_i, y_j) grid to $(\hat{x}_i, y_j)$ grid), then 1D linear interpolation in the y-direction (so $(\hat{x}_i, y_j)$ grid to $(\hat{x}_i, \hat{y}_j)$ grid). Write pseudo-code for a function that uses this method to take `gArr`, N_x, N_y, M_x, and M_y as inputs, and outputs `gHatArr`, an $M_x \times M_y$ 2D array that stores $\hat{g}(\hat{x}_i, \hat{y}_j)$, the interpolated results.

Answer:

function LINEAR_2D(a, b, c, d, `gArr`, N_x, N_y, M_x, M_y)
    $h_x \leftarrow (b-a)/(N_x-1)$
    $\hat{h}_x \leftarrow (b-a)/(M_x-1)$
    for $j \leftarrow 0, \ldots, N_y - 1$ **do**
        for $\hat{i} \leftarrow 0, \ldots, M_x - 1$ **do**
            $x_{\hat{i}} \leftarrow a + \hat{i}\hat{h}_x$
            $i \leftarrow \text{FLOOR}((x_{\hat{i}} - a)/h_x)$
            $x_i \leftarrow a + ih_x$
            $x_{i+1} \leftarrow a + (i+1)h_x$
            $\omega_x \leftarrow (x_{i+1} - x_{\hat{i}})/h_x$
            $\texttt{gHatArrTemp}[\,\hat{i}, j\,] \leftarrow \omega_x \texttt{gArr}[\,i, j\,] + (1-\omega_x)\texttt{gArr}[\,i+1, j\,]$
        end for
    end for
    $h_y \leftarrow (d-c)/(N_y-1)$
    $\hat{h}_y \leftarrow (d-c)/(M_y-1)$
    for $\hat{j} \leftarrow 0, \ldots, M_y - 1$ **do**
        for $\hat{i} \leftarrow 0, \ldots, M_x - 1$ **do**
            $y_{\hat{j}} \leftarrow c + \hat{j}\hat{h}_y$
            $j \leftarrow \text{FLOOR}((y_{\hat{j}} - c)/h_y)$
            $y_j \leftarrow c + jh_y$
            $y_{j+1} \leftarrow c + (j+1)h_y$
            $\omega_y \leftarrow (y_{j+1} - y_{\hat{j}})/h_j$
            $\texttt{gHatArr}[\,\hat{i}, \hat{j}\,] \leftarrow \omega_y \texttt{gHatArrTemp}[\,\hat{i}, j\,] + (1-\omega_y)\texttt{gHatArrTemp}[\,\hat{i}, j+1\,]$
        end for
    end for
    return `gHatArr`
end function

(c) (10 points) Rather than doing two rounds of 1D interpolation on all rows and columns as in part (b), which requires quite a bit of slow data movement, it is more efficient to work on just a couple rows of data at a time. Do linear interpolation on those two rows of data, then linearly interpolate between the two x-values represented by those two rows. Then move on to the next row. Write pseudo-code for this more efficient method.

Answer:

function LINEAR_2D(a, b, c, d, gArr, N_x, N_y, M_x, M_y)
  $h_x \leftarrow (b-a)/(N_x-1)$
  $\hat{h}_x \leftarrow (b-a)/(M_x-1)$
  $h_y \leftarrow (d-c)/(N_y-1)$
  $\hat{h}_y \leftarrow (d-c)/(M_y-1)$
  for $\hat{j} \leftarrow 0, \ldots, M_y-1$ **do**
    $y_{\hat{j}} \leftarrow c + \hat{j}\hat{h}_y$
    $j \leftarrow \text{FLOOR}((y_{\hat{j}}-c)/h_y)$
    $y_j \leftarrow c + jh_y$
    $y_{j+1} \leftarrow c + (j+1)h_y$
    $\omega_y \leftarrow (y_{j+1} - y_{\hat{j}})/h_j$
    for $\hat{i} \leftarrow 0, \ldots, M_x-1$ **do**
      $x_{\hat{i}} \leftarrow a + \hat{i}\hat{h}_x$
      $i \leftarrow \text{FLOOR}((x_{\hat{i}}-a)/h_x)$
      $x_i \leftarrow a + ih_x$
      $x_{i+1} \leftarrow a + (i+1)h_x$
      $\omega_x \leftarrow (x_{i+1} - x_{\hat{i}})/h_x$
      $\hat{g}_j \leftarrow \omega_x \texttt{gArr}[\,i,j\,] + (1-\omega_x)\texttt{gArr}[\,i+1,j\,]$
      $\hat{g}_{j+1} \leftarrow \omega_x \texttt{gArr}[\,i,j+1\,] + (1-\omega_x)\texttt{gArr}[\,i+1,j+1\,]$
      $\texttt{gHatArr}[\,\hat{i},\hat{j}\,] \leftarrow \omega_y\hat{g}_j + (1-\omega_y)\hat{g}_{j+1}$
    end for
  end for
  return gHatArr
end function

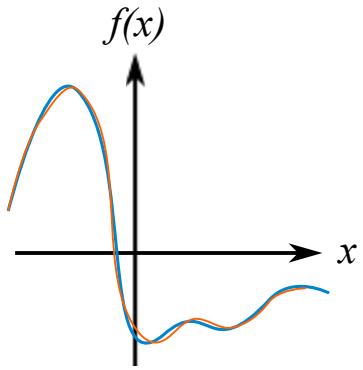


Figure 4: Cubic approximation (orange) is closer to the true function (blue) than linear interpolation.

5. **Piece-wise cubic polynomial interpolation** Using the same setup as problem 4 and 5, if one wishes to estimate $f(\hat{x})$ by $\hat{f}(\hat{x})$ where $x_i \leq \hat{x} \leq x_{i+1}$, an even more accurate approximation would be to fit a cubic polynomial on the interval $[x_i, x_{i+1}]$ based on the values of $f(x_{i-1})$, $f(x_i)$, $f(x_{i+1})$, and $f(x_{i+2})$ for $i = 1, 2, \ldots, N-2$. Assuming that we want a cubic model of the form $\hat{f}(\hat{x}) = \alpha\hat{x}^3 + \beta\hat{x}^2 + \gamma\hat{x} + \delta$ that is exactly equal to the function values at x_{i-1}, x_i, x_{i+1} and x_{i+2}, we can set up and solve a four-by-four system for α, β, γ and δ:

$$\begin{aligned}
\hat{f}(\hat{x}_{i-1}) &= \alpha\hat{x}_{i-1}^3 + \beta\hat{x}_{i-1}^2 + \gamma\hat{x}_{i-1} + \delta \\
\hat{f}(\hat{x}_i) &= \alpha\hat{x}_i^3 + \beta\hat{x}_i^2 + \gamma\hat{x}_i + \delta \\
\hat{f}(\hat{x}_{i+1}) &= \alpha\hat{x}_{i+1}^3 + \beta\hat{x}_{i+1}^2 + \gamma\hat{x}_{i+1} + \delta \\
\hat{f}(\hat{x}_{i+2}) &= \alpha\hat{x}_{i+2}^3 + \beta\hat{x}_{i+2}^2 + \gamma\hat{x}_{i+2} + \delta
\end{aligned}$$

which can be set up as a linear system:

$$\begin{bmatrix} \hat{f}(\hat{x}_{i-1}) \\ \hat{f}(\hat{x}_i) \\ \hat{f}(\hat{x}_{i+1}) \\ \hat{f}(\hat{x}_{i+2}) \end{bmatrix} = \begin{bmatrix} \hat{x}_{i-1}^3 & \hat{x}_{i-1}^2 & \hat{x}_{i-1} & 1 \\ \hat{x}_i^3 & \hat{x}_i^2 & \hat{x}_i & 1 \\ \hat{x}_{i+1}^3 & \hat{x}_{i+1}^2 & \hat{x}_{i+1} & 1 \\ \hat{x}_{i+2}^3 & \hat{x}_{i+2}^2 & \hat{x}_{i+2} & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ \delta \end{bmatrix} \tag{1}$$

When you solve this system for α, β, γ and δ, this yields a new interpolation method for $x_i \leq \hat{x} \leq x_{i+1}$, which is $\hat{f}(\hat{x}) = \alpha\hat{x}^3 + \beta\hat{x}^2 + \gamma\hat{x} + \delta$. Approximating f on a new grid of points with this formula is called *piece-wise cubic interpolation*. A true f versus the values of its piece-wise cubic interpolation function is shown in Figure 4, and we see that the cubic interpolation function is much closer than the linear piece-wise interpolation function on the same grid. Although this sketch may appear smooth at first glance, note that this interpolation scheme does not guarantee that $\hat{f}$ is differentiable.

Because we use more neighboring points, we need to make a choice on how to handle points near the edges. For $\hat{x} < x_1$, use the same $\hat{f}$ function as we would for $x_1 \leq \hat{x} \leq x_2$. For $\hat{x} > x_{N-2}$, use the same $\hat{f}$ as we would for $x_{N-3} \leq \hat{x} \leq x_{N-2}$.

(a) (13 points) Let $[a, b] = [0, 3\pi]$, and $f(x) = \sin(x)$. You know the true value of f at the points $x_i = 0 + hi$ including $x_0 = 0$ and $x_N = 3\pi$. Let $h = \frac{2\pi}{3}$. Using piece-wise cubic interpolation, estimate the value of f at $x = 3$. What is the error from the true value? Repeat this exercise with $h = \frac{\pi}{3}$. Report the values of all your α, β, γ and δ coefficients in both cases. How do the errors compare? Note, to solve the 4x4 systems, you may write code in any language (for example: existing functions in Matlab or Python with Numpy, C, C++, Java, etc...). You are expected to turn that code in.

Answer: For $h = 2\pi/3$, we have $x_{i-1} = 0$, $x_i = 2\pi/3$, $x_{i+1} = 4\pi/3$, and $x_{i+2} = 2\pi$. Solving the above system with these values gives

$$\alpha = 0.09426594$$
$$\beta = -0.88843553$$
$$\gamma = 1.86073502$$
$$\delta = 0.0$$

and an error of 0.0096544.
For $h = \pi/3$, we have $x_{i-1} = \pi/3$, $x_i = 2\pi/3$, $x_{i+1} = \pi$, and $x_{i+2} = 4\pi/3$. Solving the above system with these values gives

$$\alpha = 0.12568792$$
$$\beta = -1.18458071$$
$$\gamma = 2.75664448$$
$$\delta = -0.8660254$$

and an error of 0.00486459.
They are better (smaller) than the previous cases.
Python code to generate these results:

```
import numpy as np

# Create a 1D vector with the x_i values for h=2pi/3
xs = 2*np.pi/3 * np.arange(4)

# Create a 1D vector with the x_i values for h=pi/3
# xs = 2*np.pi/3 * np.arange(1,5)

# Make it a column vector
xs.shape = 4,1

# Create the Vandermonde matrix and RHS
A = np.hstack([xs**3, xs**2, xs**1, xs**0])
b = np.sin(xs)

#Solve the system
coeffs = np.linalg.solve(A,b)

# Define the interpolant
def f_hat(x):
    return (coeffs[0]*x**3 + coeffs[1]*x**2 +
            coeffs[2]*x**1 + coeffs[3]*x**0 )

# Define true function
f = np.sin

# Define error function
e = lambda x: np.abs(f_hat(x) - f(x))

# Evaluate error
e(3.0)
```

To interpolate onto an entirely new grid, you would need to have a for-loop, with a 4x4 system solve inside each loop.

6. (2 points) Who did you collaborate with on this assignments? What references did you use?

7. (Extra credit, 10 points) Modify your more efficient pseudocode from problem 5(c) to do piece-wise cubic interpolation instead of linear interpolation on a 2D array. *Hint*: You'll work with 4 rows of data at a time. Be sure to specify how you would treat edges.

Answer:

function CUBIC_2D(a, b, c, d, gArr, N_x, N_y, M_x, M_y)
    $h_x \leftarrow (b-a)/(N_x-1)$
    $\hat{h}_x \leftarrow (b-a)/(M_x-1)$
    $h_y \leftarrow (d-c)/(N_y-1)$
    $\hat{h}_y \leftarrow (d-c)/(M_y-1)$
    for $\hat{j} \leftarrow 0, \ldots, M_y-1$ **do**
        $y_{\hat{j}} \leftarrow c + \hat{j}\hat{h}_y$
        $j \leftarrow \text{FLOOR}((y_{\hat{j}}-c)/h_y)$
        $j \leftarrow \text{MAX}(1, j)$
        $j \leftarrow \text{MIN}(j, N_y-3)$
        $y_{j-1} \leftarrow c + (j-1)h_y$
        $y_{j+0} \leftarrow c + (j+0)h_y$
        $y_{j+1} \leftarrow c + (j+1)h_y$
        $y_{j+2} \leftarrow c + (j+2)h_y$
        for $\hat{i} \leftarrow 0, \ldots, M_x-1$ **do**
            $x_{\hat{i}} \leftarrow a + \hat{i}\hat{h}_x$
            $i \leftarrow \text{FLOOR}((x_{\hat{i}}-a)/h_x)$
            $i \leftarrow \text{MAX}(1, i)$
            $i \leftarrow \text{MIN}(i, N_x-3)$
            $x_i \leftarrow a + ih_x$
            $x_{i-1} \leftarrow a + (i-1)h_x$
            $x_{i+0} \leftarrow a + (i+0)h_x$
            $x_{i+1} \leftarrow a + (i+1)h_x$
            $x_{i+2} \leftarrow a + (i+2)h_x$
            for $k \leftarrow -1, 0, 1, 2$ **do**
                coeffs $\leftarrow$ SOLVE_COEFFICIENTS($x_{i-1}, x_{i+0}, x_{i+1}, x_{i+2}$, gArr$[i-1, j+k]$, gArr$[i, j+k]$, gArr$[i+1, j+k]$, gArr$[i+2, j+k]$)
                g$[k+1] \leftarrow$ APPLY_COEFFICIENTS(coeffs, $x_{\hat{i}}$)
            end for
            coeffs $\leftarrow$ SOLVE_COEFFICIENTS($y_{j-1}, y_{j+0}, y_{j+1}, y_{j+2}$, g$[0]$, g$[1]$, g$[2]$, g$[3]$)
            gHatArr$[\hat{i}, \hat{j}] \leftarrow$ APPLY_COEFFICIENTS(coeffs, $y_{\hat{j}}$)
        end for
    end for
    return gHatArr
end function