Problem Set 7

Author: Aaron C Watt

Date: 2022-11-08

Working with: Aidan Wang, Tak-Huen Chau, John Kadlick

Navigation of this document: the functions used in this problem set are spread out throughout the file, sited mostly where they are used to show how the data were generated. There are a few auxiary functions at the end of the file.

```
begin
    # using Pkg; Pkg.add(["Plots", "DataFrames", "PyPlot"])
    # Shouldn't need to use the above line in Pluto (automatic installation)
    using Plots ✓
    gr()
    using DataFrames ✓, Statistics ✓
    DIGITS = 4  # rounding to DIGITS for display
end;
```

Approximation of a bivariate function

$$f(x,y) = log(x+y)$$

f

```
f(x)
```

f(x) and f(x,y) for the problem

f(x) = (x-2)^(1/2) if x>=2, else 0

```
f(x,y)
```

f(x,y) = log(x+y)

```
"""
    f(x)

f(x) and f(x,y) for the problem

f(x) = (x-2)^(1/2) if x>=2, else 0

    f(x,y)

f(x,y) = log(x+y)
"""
f(x) = x<2 ? 0 : (x-2)^(1/2)
```

```
f(x,y) = log(x+y);
```

(a) The `PolyBasis` function

PolyBasis

PolyBasis(K,Z) Polynomial basis functions. Using 2nd order polynomial inputs K n x 1 points for K Z n x 1 points for Z (or scalar) outputs B n x 6 array of basis functions: 1, K, Z, K^2, K*Z, Z^2

PolyBasis(X) PolyBasis for the 1-d problem

```
"""
PolyBasis(K,Z)
    Polynomial basis functions.  Using 2nd order polynomial
    inputs
    K    n x 1   points for K
    Z    n x 1   points for Z (or scalar)
    outputs
    B    n x 6   array of basis functions: 1, K, Z, K^2, K*Z, Z^2

PolyBasis(X)
    PolyBasis for the 1-d problem
"""
function PolyBasis(X,Y)
    Yb = Y.*ones(size(X));
    B = [ones(size(X)) X Yb X.^2 X.*Yb Yb.^2];
    return B
end
```

(b) The `PolyGetCoeff` function

PolyGetCoef

```
PolyGetCoef(X,Y,Z)
```

Fits the polynomial from PolyBasis to the function(s) in column(s) of Z.

inputs

```
K    n x 1   points for K
Z    n x 1   points for Z
Y    n x 1   values for function at (X,Y)
```

outputs

```
b    6 x 1   basis coefficients
```

```
"""
    PolyGetCoef(X,Y,Z)

Fits the polynomial from PolyBasis to the function(s) in column(s) of Z.

inputs

    K    n x 1   points for K
    Z    n x 1   points for Z
    Y    n x 1   values for function at (X,Y)

outputs

    b    6 x 1   basis coefficients
"""
function PolyGetCoef(X,Y,Z)
    B = PolyBasis(X,Y)
    b = B \ Z;
    return b
end
```

(c)-(e) Get Polynomial Approximation Coefficients

- Create the X-Y grid
- Calculate true f(X,Y)
- Use `PolyGetCoef` to calculate polynomial approximation coefficients

gen_2d_coef

```
gen_2d_coef(n)
```

Generate the polynomial basis coefficients for the 2-d function, where `n` is the number of points used in the equally-spaced range of numbers to sample from the true function.

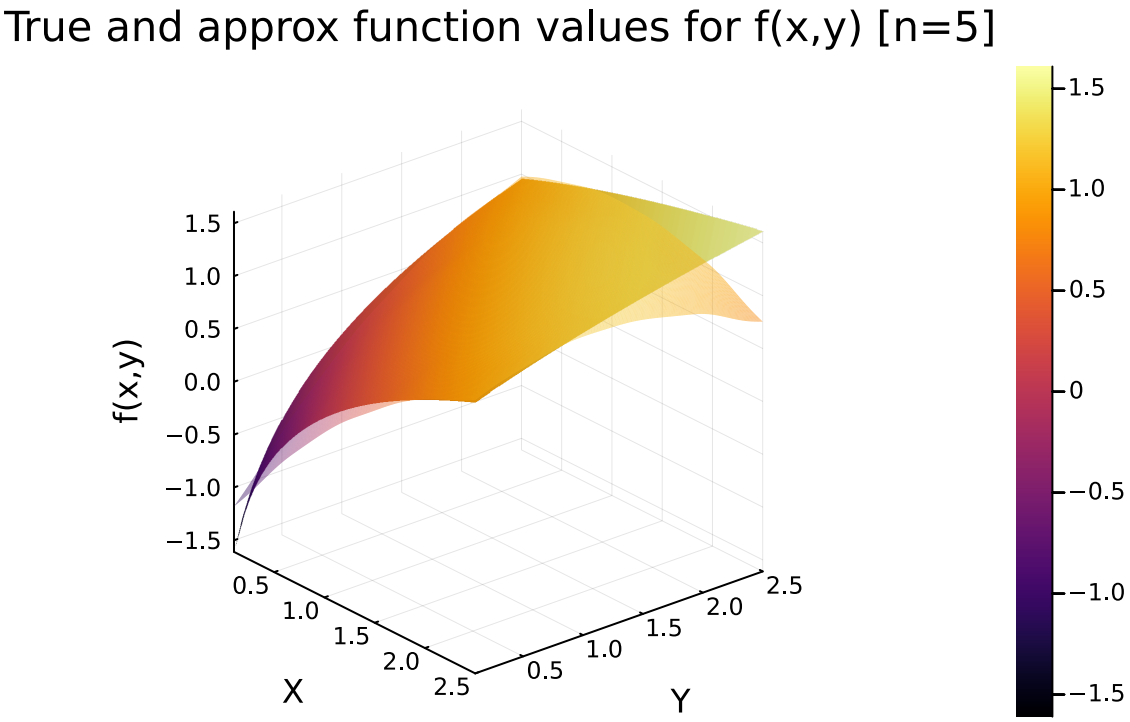
```
"""
    gen_2d_coef(n)

Generate the polynomial basis coefficients for the 2-d function, where `n` is the
number of points used in the equally-spaced range of numbers to sample from the
true function.
"""
function gen_2d_coef(n)
    START=0.1; STOP=2.0
    X = range(START,STOP,n)
    Y = range(START,STOP,n)
    XY = [(x,y) for x ∈ X for y ∈ Y]
    XX = [x for (x,y) ∈ XY]
    YY = [y for (x,y) ∈ XY]
    Z = [log(x+y) for (x,y) ∈ XY]
    b = PolyGetCoef(XX,YY,Z)
    return b
end
```

(f)-(g) Calculate the true and approximated function values on a larger grid

- repeating for n=15 and 35 to use in part (k)
- also calcuating the error norm to use in part (i)

```
begin
    # Generate the approximated values and error norms for the 2-d problem for
    n=5,15,35
    data_2d = Dict()
    for n ∈ [5, 15, 35]
        # Generate polynomial coefficients on the short grid (0.1-2.0)
        b = gen_2d_coef(n)
        # Calculate the error norm on the short grid
        _, _, _, _, norm_in = gen_2d_extrapolation(n, b; STOP=2.0)
        # Calculate true and approx function values on extended grid (0.1-2.5)
        (X,Y), Z, Zhat, _, norm_out = gen_2d_extrapolation(n, b; STOP=2.5)
        data_2d[n] = Dict("X"=>X, "Y"=>Y, "Z"=>Z, "Zhat"=>Zhat,
            "norm in"=>norm_in, "norm out"=>norm_out)
    end
end
```

(h) Plot the true and approximated surfaces



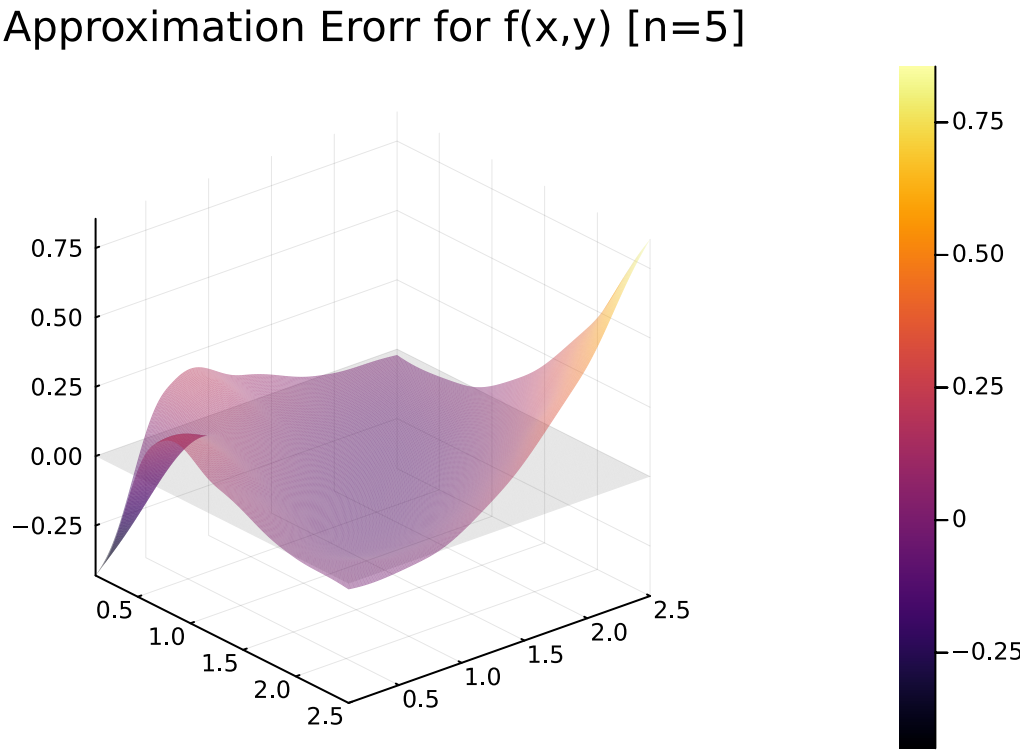

```
plot_surface(5)
```

(i) Maximum absolute value of the difference

Max Interpolation Error = 0.4302 (over the range 0.1-2.0)

Max Extrapolation Error = 0.8553 (over the range 0.1-2.5)

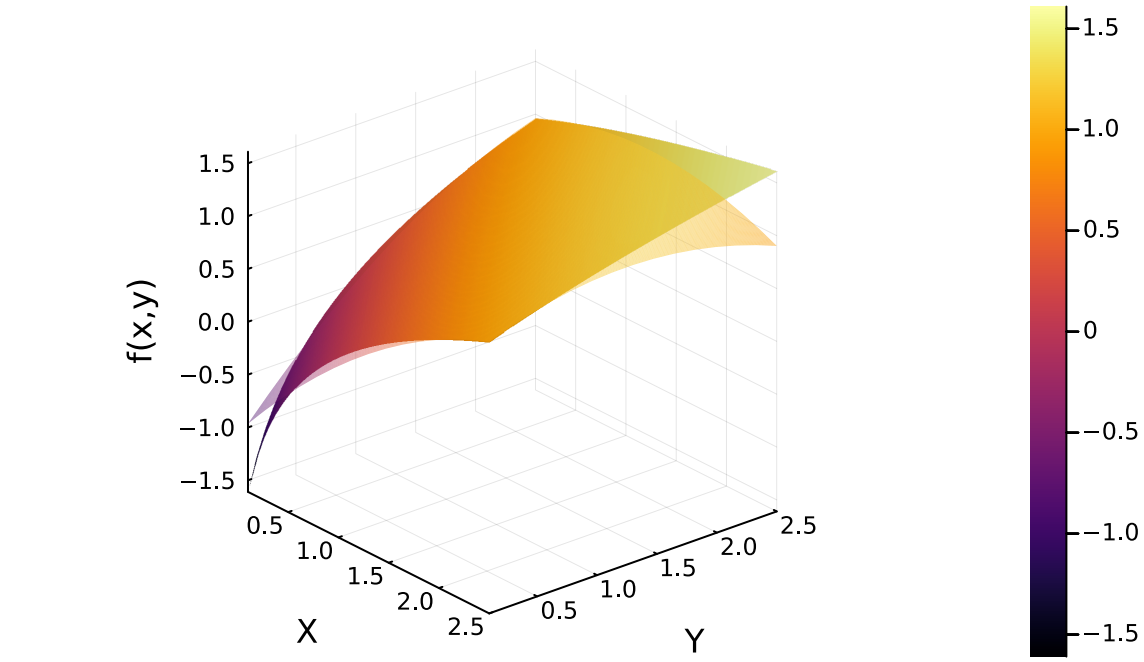
(j) Plot the difference between the true and approximated surfaces




```
begin
    # Use local variables because Pluto Notebooks can only define globals once
    local x = data_2d[5]["X"]; local y = data_2d[5]["Y"];
    local ε = data_2d[5]["Z"] .- data_2d[5]["Zhat"];
    # Plot the surface and a 0 plane
    surface(x, y, zeros(size(x)), color=cgrad([:grey,:grey]), alpha=0.2)
    surface!(x, y, ε,
        title="Approximation Erorr for f(x,y) [n=$n]",
        camera=(50,25),alpha = 0.5)
end
```

(k) Repeat for n=15 and 35

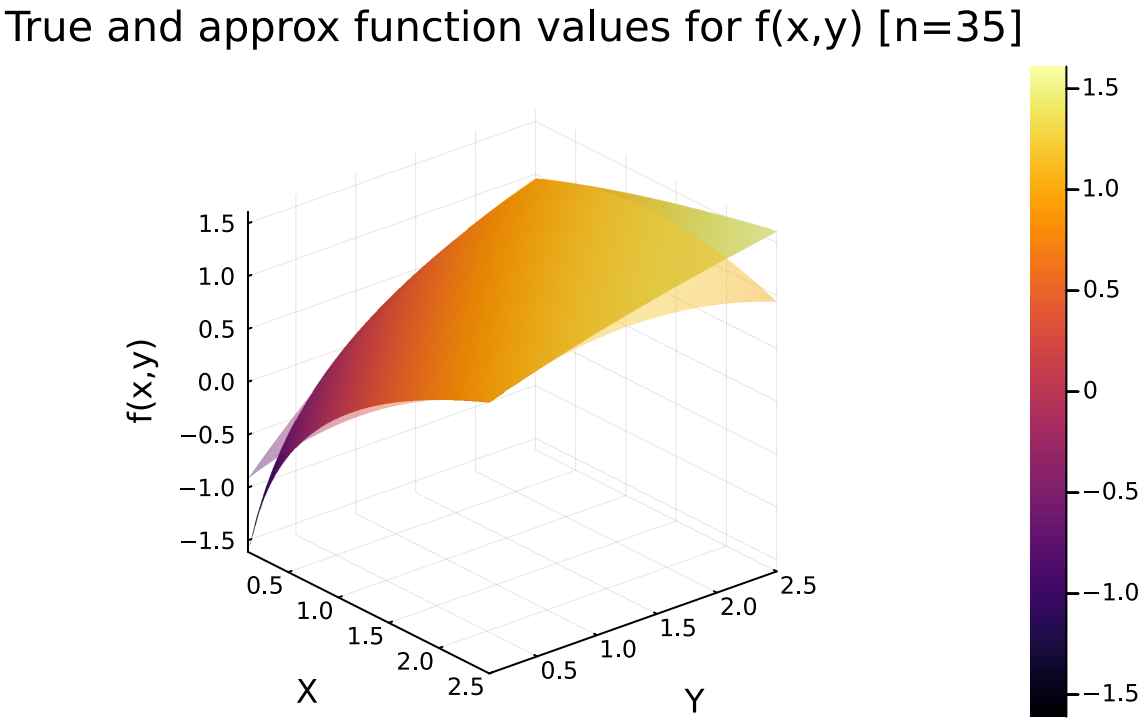
- code for generating n=15 and n=35 data is in section (f)
- plotting on different plots because 3-D plots are hard to see when there's more than 2




```
plot_surface(15)
```

Max Interpolation Error = 0.6452

Max Extrapolation Error = 0.7062




```
plot_surface(35)
```

Max Interpolation Error = 0.6931

Max Extrapolation Error = 0.6931

These are the same because the max abs. error occurs within the interpolated range (probably around 0.1).

(l) Discuss the quality of polynomial approximations

The approximations tend to do well in the middle of the range used to generate the coefficients and do the worse near the edges of the range. They do worse as we leave the sample and extrapolate. As we increase the number of points used to estimate the coefficients (`n`), we see that the maximum absolute error (MAE) of the approximations in the extrapolated range of points decreases, but the MAE of the approximations increases in the interpolated range.

I think this makese sense because the MAE in the smaller range used to estimate the quadratic coefficients seems to be driven by the error at 0.1 (the lower end of the range). This is because `log` is so extreme near 0. When we increase the number of points used inside the small range (0.1 to 2.0), then we are putting more weight on the middle of the range and less weight on the edges. So if the MAE is largest for the interpolation at the lower end of the range, but it gets less weight when we increase the density of points, then the quadratic would fit less well there and lead to increasing MAE with increasing `n`.

(m) Approximation of a univariate function

$$f(x) = \begin{cases} 0 & x < 2 \\ (x-2)^{1/2} & x \geq 2 \end{cases}$$

Functions for the univarite problem

Note: We can use functions of the same name but with different arguments – this is multiple dispatch and is common in Julia. The functions must have either a different number of arguments or different types of arguments. Since `gen_2d_vectors` and `gen_1d_vectors` both only take `n`, I needed to create different functions.

```
Enter cell code...
```

```
function PolyBasis(X)
    B = [ones(size(X)) X X.^2]
    return B
end;
```

```
function PolyGetCoef(X,Z)
    B = PolyBasis(X)
    b = B \ Z;
    return b
end;
```

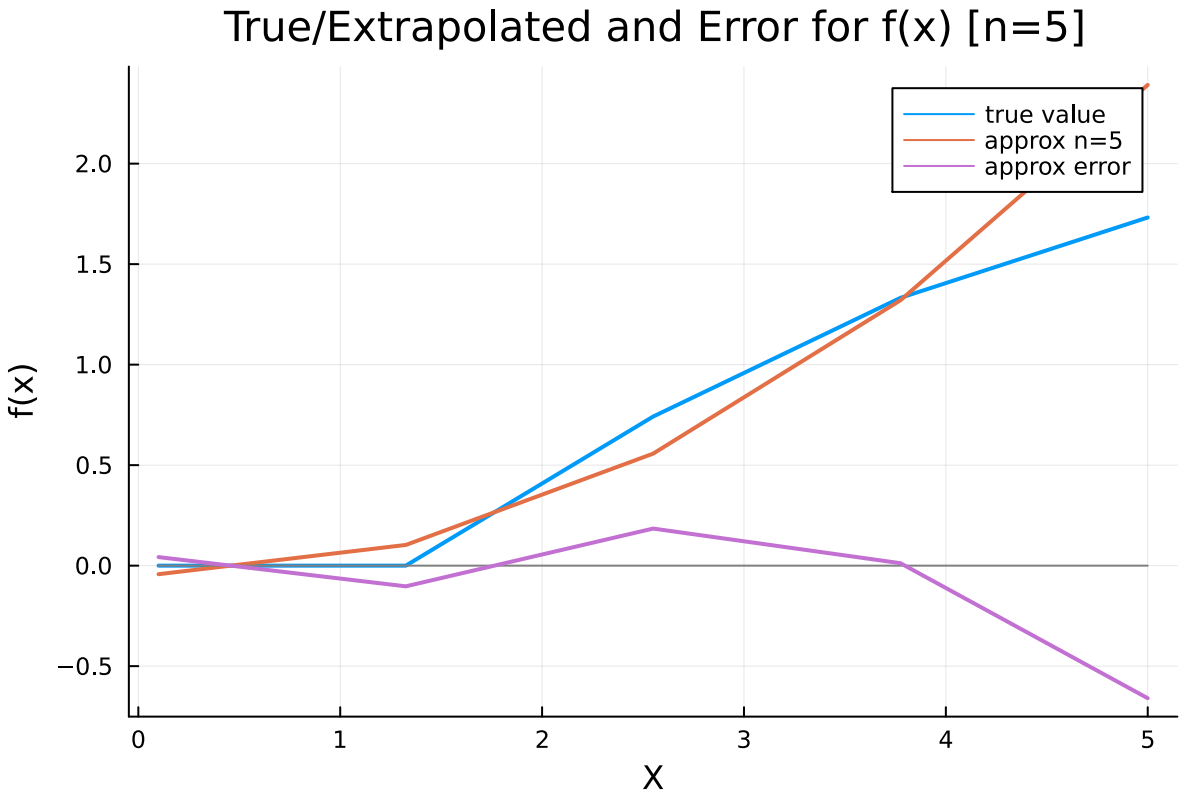
gen_1d_coef

```
gen_1d_coef(n)
```

Generate the polynomial basis coefficients for the 1-d function, where `n` is the number of points used in the equally-spaced range of numbers to sample from the true function.

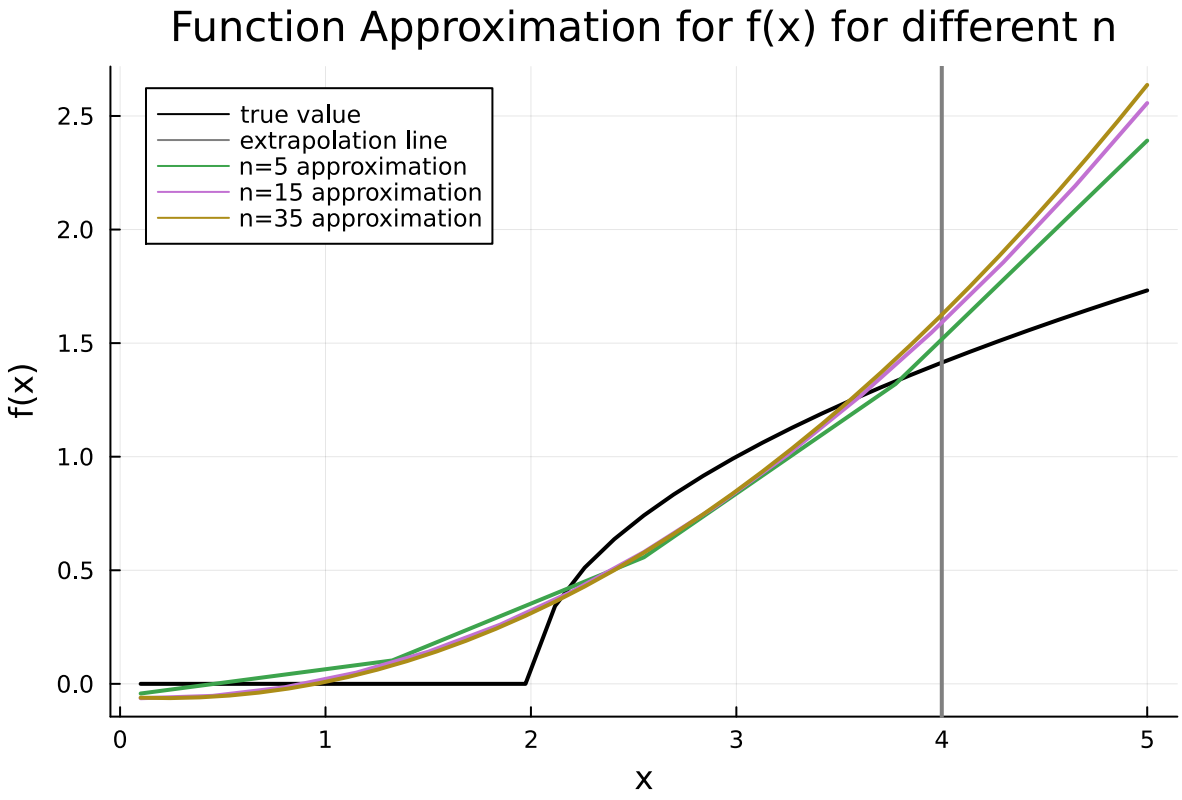
```
"""
    gen_1d_coef(n)

Generate the polynomial basis coefficients for the 1-d function, where `n` is the
number of points used in the equally-spaced range of numbers to sample from the
true function.
"""
function gen_1d_coef(n)
    START=0.1; STOP=4.0
    X = range(START, STOP, n)
    Z = [f(x) for x ∈ X]
    b = PolyGetCoef(X,Z)
    return b
end
```

Plotting for the n=5 case




```
begin
    # Plot n=5 true-function, approx-function, and error values
    # Generate polynomial coefficients and return true function values
    n=5
    b = gen_1d_coef(n)
    # Generate extrapolation
    X, Z, Zhat, ε, norm = gen_1d_extrapolation(n, b; STOP=5)
    # Plot true function, approximate, and error
    plot_stuff(X..., Z, Zhat, ε, n)
end
```

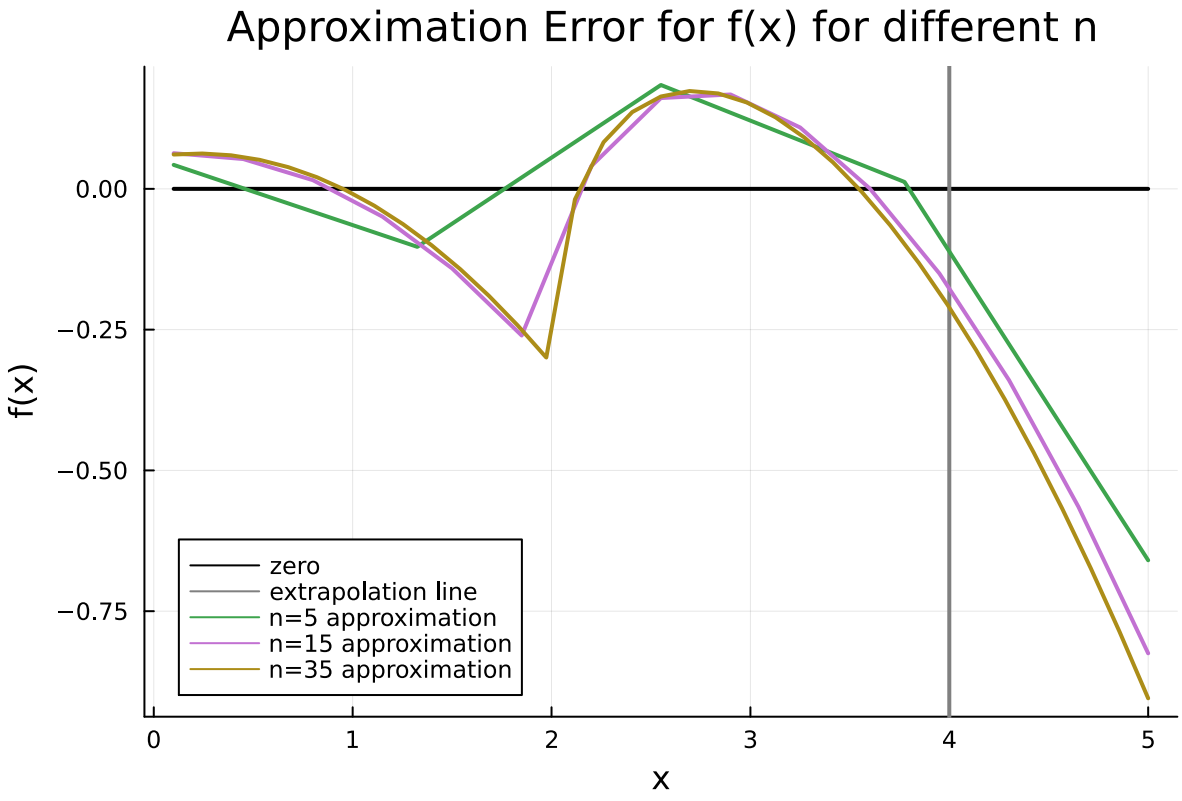
Plotting for all n cases




```
begin
    data_1d = Dict()
    for n ∈ [5, 15, 35]
        # Generate polynomial coefficients
        b = gen_1d_coef(n)
        # Generate interpolation and extrapolation, errors and error-norms
        _, _, _, _, norm_in = gen_1d_extrapolation(n, b; STOP=4.0)
        (X,), Z, Zhat, ε, norm_out = gen_1d_extrapolation(n, b; STOP=5.0)
        data_1d[n] = Dict("X"=>X, "Z"=>Z, "Zhat"=>Zhat, "norm_in"=>norm_in,
            "norm_out"=> norm_out, "error"=>ε)
    end
    # Get most detailed "true function values" from the last n=35 set of data
    p_1d = plot(data_1d[35]["X"], data_1d[35]["Z"], lw=2, color=:black,
         label="true value",
         title="Function Approximation for f(x) for different n",
         legend=:topleft,
         xlabel="x", ylabel="f(x)")
    # plot extrapolation line
    vline!([4], lw=2, color=:grey, label="extrapolation line")
    # Add the plots for the different levels of approximation
    for n ∈ [5, 15, 35]
        p_1d = plot!(data_1d[n]["X"], data_1d[n]["Zhat"], lw=2,
                  label="n=$n approximation")
    end
    p_1d
end
```

Plotting Errors for all n cases

The vertical line at x=4.0 represents the interpolation vs extrapolation erorrs – extrapolation occurs to the right of the vertical line.




```
begin
    p_1d_error = plot(data_1d[35]["X"], zeros(size(data_1d[35]["X"])),
        lw=2, color=:black,
        label="zero",
        title="Approximation Error for f(x) for different n",
        legend=:bottomleft,
        xlabel="x", ylabel="f(x)")
    # plot extrapolation line
    vline!([4], lw=2, color=:grey, label="extrapolation line")
    # Add the plots for the different levels of approximation
    for n ∈ [5, 15, 35]
        plot!(data_1d[n]["X"], data_1d[n]["error"], lw=2,
                    label="n=$n approximation")
    end
    p_1d_error
end
```

Max Absolute Approximation Error

Extrapolation error: error calculated over extrapolated function values, outside of the domain of original coefficient fitting.

Interpolation error: error calculated over interpolated function values, inside of the domain of original coefficient fitting.

	Extrapolation Max Abs Error	Interpolation Max Abs Error	n
1	0.659602	0.19606	5
2	0.824819	0.231328	15
3	0.904701	0.283903	35

```
begin
    norm_ins = [data_1d[n]["norm_in"] for n ∈ [5, 15, 35]]
    norm_outs = [data_1d[n]["norm_out"] for n ∈ [5, 15, 35]]
    DataFrame(Dict("n"=>[5, 15, 35],
        "Interpolation Max Abs Error"=>norm_ins,
        "Extrapolation Max Abs Error"=>norm_outs))
end
```

	Extrapolation Mean Abs Error	Interpolation Mean Abs Error	n
1	0.200358	0.200358	5
2	0.196082	0.196082	15
3	0.199873	0.199873	35

```
begin
    mean_ins2 = [mean(abs.(data_1d[n]["error"])) for n ∈ [5, 15, 35]]
    mean_outs2 = [mean(abs.(data_1d[n]["error"])) for n ∈ [5, 15, 35]]
    DataFrame(Dict("n"=>[5, 15, 35],
        "Interpolation Mean Abs Error"=>mean_ins2,
        "Extrapolation Mean Abs Error"=>mean_outs2))
end
```

Discussion for the univariate function

From top table above, we see that the lowest n gives the lowest MAE for both interpolation and extrapolation. However, then second table shows the Mean Absolute Error – from this we see that all the n's seem to perform very closesly on average over their range, with the 15 and 35 cases beign slightly closer to the true values than $n=15$. The mean values are also fairly close to the interpolation MAE, but the extrapolation MAE are much larger. This tells me that extrapolation from this polynomial fit can produce very large prediction errors resulting from model mispecification.

Auxiliary Functions

gen_2d_extrapolation

```
gen_2d_extrapolation(n, b; START=0.1, STOP=2.5)
```

Generate the X, Y, and Zhat arrays from the extrapolation over the array of `n` values in the domain of X,Y (ranging from `START` to `STOP`), extrapolating using the previously fit polynomial basis represented by `b`.

inputs:

```
n = number of points used
b = vector of polynomial basis coefficients
START = starting number for X and Y arrays
STOP = stopping number for X and Y arrays
```

outputs:

```
XX = array of n equally-spaced X points, repeated (inner) n times for plot
ting
YY = array of n equally-spaced Y points, repeated (outer) n times for plot
ting
Z = array of n function values = f(X)
Zhat = array of n approximated function values using the basis coefficient
s b
ε = array of n errors = Z - Zhat
norm = maximum of the absolute value of the errors
```

```
"""
    gen_2d_extrapolation(n, b; START=0.1, STOP=2.5)

Generate the X, Y, and Zhat arrays from the extrapolation over the array of `n`
values in the domain of X,Y (ranging from `START` to `STOP`), extrapolating using
the previously fit polynomial basis represented by `b`.

inputs:

    n = number of points used
    b = vector of polynomial basis coefficients
    START = starting number for X and Y arrays
    STOP = stopping number for X and Y arrays

outputs:

    XX = array of n equally-spaced X points, repeated (inner) n times for plotting
    YY = array of n equally-spaced Y points, repeated (outer) n times for plotting
```

```
    Z = array of n function values = f(X)
    Zhat = array of n approximated function values using the basis coefficients b
    ε = array of n errors = Z - Zhat
    norm = maximum of the absolute value of the errors
"""
function gen_2d_extrapolation(n, b; START=0.1, STOP=2.5)
    X = range(START,STOP,n)
    Y = range(START,STOP,n)
    XY = [(x,y) for x ∈ X for y ∈ Y]
    XX = [x for (x,y) ∈ XY]
    YY = [y for (x,y) ∈ XY]
    Z = [log(x+y) for (x,y) ∈ XY]
    B = PolyBasis(XX,YY)
    Zhat = B*b
    ε = Z - Zhat
    norm = maximum(abs.(ε))
    return (XX,YY), Z, Zhat, ε, norm
end
```

gen_1d_extrapolation

```
gen_1d_extrapolation(n, b; START=0.1, STOP=2.5)
```

Generate the X and Zhat arrays from the extrapolation over the array of `n` values in the domain of X (ranging from `START` to `STOP`), extrapolating using the previously fit polynomial basis represented by `b`.

inputs:

```
n = number of points in the equally-spaced array of X points
b = vector of polynomial basis coefficients
START = starting number for X and Y arrays
STOP = stopping number for X and Y arrays
```

outputs:

```
X = array of n equally-spaced X points
Z = array of n function values = f(X)
Zhat = array of n approximated function values using the basis coefficient
s b
ε = array of n errors = Z - Zhat
norm = maximum of the absolute value of the errors
```

```
"""
    gen_1d_extrapolation(n, b; START=0.1, STOP=2.5)

Generate the X and Zhat arrays from the extrapolation over the array of `n`
values in the domain of X (ranging from `START` to `STOP`), extrapolating using
the previously fit polynomial basis represented by `b`.

inputs:

    n = number of points in the equally-spaced array of X points
    b = vector of polynomial basis coefficients
    START = starting number for X and Y arrays
    STOP = stopping number for X and Y arrays

outputs:

    X = array of n equally-spaced X points
    Z = array of n function values = f(X)
    Zhat = array of n approximated function values using the basis coefficients b
    ε = array of n errors = Z - Zhat
    norm = maximum of the absolute value of the errors
```

```
"""
function gen_1d_extrapolation(n, b; START=0.1, STOP=4.5)
    X = range(START,STOP,n)
    Z = [f(x) for x ∈ X]
    B = PolyBasis(X)
    Zhat = B*b
    ε = Z - Zhat
    norm = maximum(abs.(ε))
    return (X,), Z, Zhat, ε, norm
end
```

plot_stuff

```
plot_stuff(X, Z, Zhat, ε, n)
```

Plot the true function values, the approximated function values, and the error on the same plot, for the 1-d problem.

```
"""
    plot_stuff(X, Z, Zhat, ε, n)

Plot the true function values, the approximated function values, and the error on
the same plot, for the 1-d problem.
"""
function plot_stuff(X, Z, Zhat, ε, n)
    plot(X, Z, label="true value", lw=2)
    plot!(X, Zhat, label="approx n=$n", lw=2)
    plot!(X, zeros(size(X)), label="", color=:grey)
    plot!(X, ε, label="approx error",
        title="True/Extrapolated and Error for f(x) [n=$n]", lw=2,
        xlabel="X", ylabel="f(x)")
end
```

plot_surface

```
plot_surface(n)
```

Plot the true function values, the approximated function values, and the error on the same plot, for the 2-d problem for n=`n`.

```
"""
    plot_surface(n)

Plot the true function values, the approximated function values, and the error on
the same plot, for the 2-d problem for n=`n`.
"""
function plot_surface(n)
    surface(data_2d[35]["X"], data_2d[35]["Y"], data_2d[35]["Z"])
    s = surface!(data_2d[n]["X"], data_2d[n]["Y"], data_2d[n]["Zhat"],
        title="True and approx function values for f(x,y) [n=$n]",
        camera=(50,25), alpha = 0.5,
        xlabel="X", ylabel="Y", zlabel="f(x,y)")
    return s
end
```

Appendix - Unused code, saved for later

```
# begin
#   # depreciated - used to print all plots for both 1d and 2d problems
#   plots = []
#   for n ∈ [5, 15, 35]
#       for dimensions in 1:2
#           println("n=$n, dim=$dimensions")
#           if dimensions == 1  # select 1 dimensional problem
#               gen_coef = gen_1d_coef
#               gen_extrap = gen_1d_extrapolation
#               stop = 5
#           elseif dimensions == 2  # select 2 dimensional problem
#               gen_coef = gen_1d_coef
#               gen_extrap = gen_1d_extrapolation
#               stop = 2.5
#           end
#           # Generate polynomial coefficients and return true function values
#           b = gen_coef(n)
#           # Generate extrapolation
#           X, Z, Zhat, ε, norm = gen_extrap(n, b; STOP=stop)
#           # Plot true function, approximate, and error
#           plot_stuff(X..., Z, Zhat, ε, n)
#       end
#   end
# end
```

```
# begin
#   # depreciated - used to try to plot all 3d plots, but needed to plot
separately
#   data_2d = Dict()
#   for n ∈ [5, 15, 35]
#       # Generate polynomial coefficients and return true function values
#       b = gen_2d_coef(n)
#       # Generate extrapolation
#       X, Z, Zhat, _, _ = gen_2d_extrapolation(n, b; STOP=stop)
#       data_2d[n] = Dict("X"=>X, "Z"=>Z, "Zhat"=>Zhat)
#   end
#   # Get most detailed "true function values" from the last n=35 set of data
#   p_2d = plot(data_2d[35]["X"], data_2d[35]["Z"], lw=2, color=:black,
#        label="true value",
#        title="Function Approximation for f(x)",
#        legend=:bottomright,
#        xlabel="x", ylabel="f(x)")
#   # Add the plots for the different levels of approximation
#   for n ∈ [5, 15, 35]
#       p_2d = plot!(data_2d[n]["X"], data_2d[n]["Zhat"], lw=2,
#                    label="n=$n approximation")
#   end
#   p_2d
# end
```

```
begin
    # Create a table of contents sidebar to help with navigation
    # Comment out when creating a PDF
    #### Pkg.add("PlutoUI")
    # using PlutoUI
    # TableOfContents()
end
```