```julia
# ------------------------------------------------------------------------------
# Author: Philip Coyle
# Date Created: 06/10/2020
# ps2.jl
# ------------------------------------------------------------------------------

using Optim, Plots, Interpolations, Statistics
dir = "/Users/philipcoyle/Documents/School/University_of_Wisconsin/SecondYear/Summer_2020/CodingBootcamp/ProblemSets/PS2/"

## Question 1
function Himmelblau(in::Array{Float64,1})
    x = in[1]
    y = in[2]

    out = (x^2 + y - 11)^2 + (x + y^2 - 7)^2
    return out
end

function g(G, in::Array{Float64,1})
    x = in[1]
    y = in[2]

    G[1] = (4*x)*(x^2 + y - 11) + 2*(x + y^2 - 7)
    G[2] = 2*(x^2 + y - 11) + (4*y)*(x + y^2 - 7)
end
function h(H, in::Array{Float64,1})
    x = in[1]
    y = in[2]

    H[1,1] = 12*x^2 + 4*y - 44 + 2
    H[1,2] = 4*x + 4*y
    H[2,1] = 4*x + 4*y
    H[2,2] = 2 + 4x + 12y - 28
end

# Part A
n_grid = 101
x_grid = range(-4, 4, length = n_grid)
y_grid = x_grid

# Preallocate space for functuon
z_grid = zeros(n_grid, n_grid)
for i in 1:n_grid
    for j = 1:n_grid
        vec_in = [x_grid[i], y_grid[j]]
        z_grid[i,j] = Himmelblau(vec_in)
    end
end

# Part B (Trying different guesses: argmin after convergence)
n_grid = 101

for i in 1:n_grid
    for j = 1:n_grid
        guess = [x_grid[i], y_grid[j]]
        opt_newton = optimize(Himmelblau, g, h, guess)
        if opt_newton.minimizer[1] < 0
            minimizer[i,j] = 3
            if opt_newton.minimizer[2] < 0
                minimizer[i,j] = 4
            end
        else
            minimizer[i,j] = 2
            if opt_newton.minimizer[2] > 0
                minimizer[i,j] = 1
            end
        end
    end
end

# Part C (Trying different guesses: Number of iterations)
num_iter_newton = zeros(n_grid, n_grid)
num_iter_nm = zeros(n_grid, n_grid)
for i in 1:n_grid
    for j = 1:n_grid
        guess = [x_grid[i], y_grid[j]]

        opt_newton = optimize(Himmelblau, g, h, guess)
        opt_nm = optimize(Himmelblau, guess, NelderMead())

        num_iter_newton[i,j] = opt_newton.iterations
        num_iter_nm[i,j] = opt_nm.iterations
    end
end

# Plot
p1 = surface(x_grid, y_grid, z_grid, camera = (50, 50), title="Himmelblau Function");
p2 = contourf(x_grid, y_grid, log.(z_grid), title="Contour Plot");
```

```
p3 = contourf(x_grid, y_grid, minimizer, title="Convergence Destination \n 1 ≡ [3, 2]; 2 ≡ [3.6, -1.8]; 3 ≡ [-2.8, 3.1]; 4 ≡ [-3.8, -3.3]");
p4 = contourf(x_grid, y_grid, log.(num_iter_newton), title="(Log) Numer of Iters to Converge \n Newton's Method");
p5 = contourf(x_grid, y_grid, log.(num_iter_nm ), title="(Log) Numer of Iters to Converge \n Nelder Mead");

l = @layout [a{0.5w} b{0.5w}; c{0.33h}; d{0.5w} e{0.5w}]
plot(p1, p2, p3, p4, p5, layout = l, size = (800,900))
savefig(dir * "Q1.pdf")

## Question 2
function Ackley(in::Array{Float64,1})
    x = in[1]
    y = in[2]

    out = -20*exp(-0.2*(0.5*(x^2 + y^2))^(0.5)) - exp(0.5*(cos(2*pi*x) + cos(2*pi*y))) + exp(1) + 20
    return out
end

# Part A
n_grid = 101
x_grid = range(-4, 4, length = n_grid)
y_grid = x_grid

# Preallocate space for functuon
z_grid = zeros(n_grid, n_grid)
for i in 1:n_grid
    for j = 1:n_grid
        vec_in = [x_grid[i], y_grid[j]]
        z_grid[i,j] = Ackley(vec_in)
    end
end

# Plot
p1 = surface(x_grid, y_grid, z_grid, camera = (50, 50), title="Ackley Function");
p2 = contourf(x_grid, y_grid, z_grid, title="Contour Plot");
plot(p1, p2, layout=(1,2),size = (600,250))
savefig(dir * "Q2_a.pdf")

# Part B (trying different guesses)
minimizer = zeros(n_grid, n_grid,2)
num_iter = zeros(n_grid, n_grid,2)
for i = 1:n_grid
    for j = 1:n_grid
        guess = [x_grid[i], y_grid[j]]
        opt_nm = optimize(Ackley, guess, NelderMead())
        minimizer[i,j,1] = opt_nm.minimum
        num_iter[i,j,1] = opt_nm.iterations

        opt_lbfgs = optimize(Ackley, guess, LBFGS())
        minimizer[i,j,2] = opt_lbfgs.minimum
        num_iter[i,j,2] = opt_lbfgs.iterations
    end
end

# Plot
p11 = contourf(x_grid, y_grid, log.(minimizer[:,:,1]), title="Nelder Mead \n (Log) Convergence Destination ");
p12 = contourf(x_grid, y_grid, log.(minimizer[:,:,2]), title="LBFGS \n (Log) Convergence Destination");
p21 = contourf(x_grid, y_grid, log.(num_iter[:,:,1]), title="(Log) Numer of Iters to Converge");
p22 = contourf(x_grid, y_grid, log.(num_iter[:,:,2]), title="(Log) Numer of Iters to Converge");
plot(p11, p12, p21, p22, layout=(2,2),size = (1200,950))
savefig(dir * "Q2_b.pdf")

## Question 3
function Rastrigin(x::Array{Float64,1}; A = 10)
    sum = 0
    for i = 1:length(x)
        sum = sum + x[i]^2 - A*cos(2*pi*x[i])
    end

    n = length(x)
    out = A*n + sum

    return out
end

# Part A
n_grid = 101
x_grid = range(-5.12, 5.12, length = n_grid)

# Preallocate space for functuon
z_grid_1d = zeros(n_grid,1)
for i in 1:n_grid
    z_grid_1d[i] = Rastrigin([x_grid[i]])
end

plot(x_grid, z_grid_1d, title = "Rastrigin Function")
savefig(dir * "Q3_a.pdf")

# Part B
```

```julia
n_grid = 101
x_grid = range(-5.12, 5.12, length = n_grid)

# Preallocate space for functuon
z_grid_2d = zeros(n_grid,n_grid)
for i in 1:n_grid
    for j = 1:n_grid
        x_in = [x_grid[i], x_grid[j]]
        z_grid_2d[i,j] = Rastrigin(x_in)
    end
end

p1 = surface(x_grid,x_grid,z_grid_2d, camera = (50, 50), title="Rastrigin Function");
p2 = contourf(x_grid,x_grid,log.(z_grid_2d .+ 1), title="Contour Plot");
plot(p1, p2, layout=(1,2),size = (600,250))
savefig(dir * "Q3_b.pdf")

# Part C (trying different guesses)
minimizer = zeros(n_grid, n_grid,2)
num_iter = zeros(n_grid, n_grid,2)
for i = 1:n_grid
    for j = 1:n_grid
        guess = [x_grid[i], x_grid[j]]
        opt_nm = optimize(Rastrigin, guess, NelderMead())
        minimizer[i,j,1] = opt_nm.minimum
        num_iter[i,j,1] = opt_nm.iterations

        opt_lbfgs = optimize(Rastrigin, guess, LBFGS())
        minimizer[i,j,2] = opt_lbfgs.minimum
        num_iter[i,j,2] = opt_lbfgs.iterations
    end
end

# Plot
p11 = contourf(x_grid, x_grid, log.(minimizer[:,:,1]), title="Nelder Mead \n (Log) Convergence Destination ");
p12 = contourf(x_grid, x_grid, log.(minimizer[:,:,2]), title="LBFGS \n (Log) Convergence Destination");
p21 = contourf(x_grid, x_grid, log.(num_iter[:,:,1]), title="(Log) Numer of Iters to Converge");
p22 = contourf(x_grid, x_grid, log.(num_iter[:,:,2]), title="(Log) Numer of Iters to Converge");
plot(p11, p12, p21, p22, layout=(2,2),size = (1200,950))
savefig(dir * "Q3_c.pdf")

## Question 4
function lin_interp(f::Function, a::Float64, b:: Float64, n::Int64, val::Float64)
    if val < a || val > b
        msg = "x must be betwen a and b"
        throw(msg)
    end

    if n < 1
        msg = "n must be a postive integer"
        throw(msg)
    end

    grid = range(a, b, length = n)
    f_grid = f.(grid)
    f_interp = interpolate(f_grid,BSpline(Linear()))

    # Get Index
    inx_upper = findfirst(x->x>val, grid)
    inx_lower = inx_upper - 1
    val_upper, val_lower = grid[inx_upper], grid[inx_lower]
    inx = inx_lower + (val - val_lower) / (val_upper - val_lower)

    out = f_interp(inx)
    return out
end

func(x) = log(x)
a = 1.
b = 10.
x = 1.25
n = 5

lin_interp(func, a, b, n, x)

## Question 5
function get_inx(val::Float64, grid::Array{Float64,1})
    if val >= grid[length(grid)]
        inx = length(grid)
    elseif val <= grid[1]
        inx = 1
    else
        inx_upper = findfirst(x->x>val, grid)
        inx_lower = inx_upper - 1
        val_upper, val_lower = grid[inx_upper], grid[inx_lower]
        inx = inx_lower + (val - val_lower) / (val_upper - val_lower)
    end
end
```

```julia

function approx_error(f::Function, grid::Array{Float64,1}, grid_fine::Array{Float64,1})
    f_grid = f.(grid)
    f_interp = interpolate(f_grid,BSpline(Linear()))
    f_apx = zeros(length(grid_fine),1)

    for (i, grid_i) = enumerate(grid_fine)
        grid_inx = get_inx(grid_i, grid)
        f_apx[i] = f_interp(grid_inx)
    end

    f_out = abs.(f_apx .- f.(grid_fine))
    f_out_avg = mean(f_out)

    return f_out, f_out_avg
end



function opt_grid(grid::Array{Float64,1})
    grid[1] = 0
    grid[end] =100

    f(x) = log(x+1)
    grid_fine = collect(0:0.1:100)

    ~, out = approx_error(f, grid, grid_fine)

    return out
end


x_grid = collect(range(0,100, length = 11))
x_grid_fine = collect(0:0.1:100)
f(x) = log(x+1)

# Part A
error, avg_error = approx_error(f, x_grid, x_grid_fine)
plot(x_grid, f.(x_grid), label = "Interpolated Function", color=:blue,lw = 2)
plot!(x_grid_fine, f.(x_grid_fine), label = "True Function",  color=:black,lw = 2)
plot!(x_grid_fine, error, label = "Approximation Error",  color=:red,lw = 2)
savefig(dir * "Q5_LinearGrid.pdf")


# Part C
opt = optimize(opt_grid, x_grid)
error, avg_error = approx_error(f, opt.minimizer, x_grid_fine)
plot(opt.minimizer, f.(opt.minimizer), label = "Interpolated Function", color=:blue,lw = 2)
plot!(x_grid_fine, f.(x_grid_fine), label = "True Function",  color=:black,lw = 2)
plot!(x_grid_fine, error, label = "Approximation Error",  color=:red,lw = 2)
savefig(dir * "Q5_OptimizedGrid.pdf")
```