Astronomy in Julia – ESI Optimizations

Christopher Figueroa, Sai Teja Muliki

***Abstract*—** The purpose of this project is to utilize the Julia language to analyze a sample of NASA's Exo-Planet and make a comparison between 4 variants of the ESI (Earth Similarity Index) equation.

***Index Terms*—** ESI (Earth Similarity Index), Transit, Stellar Flux, Standard ESI, Weighted ESI, Red Dwarf Star, Tidally Locked, Main Sequence Star, Custom ESI, Revised ESI, Julia, Jupyter Notebook, Linear Algebra, Plots, Iterative, XLSL, Iterative ESI, Neural Network

I. Introduction

The field of astronomy has a vast history that dates back to the first millennium BCE. The scope of research in the field remained confined to the planets within our solar system, throughout most of history. However, a significant turning point came in 1992 with the revelation of two planets orbiting the pulsar PSR 1256+12, marking the beginning of a new era in astronomy. This discovery was just the first of many thousands to come in the following years, as astronomers made groundbreaking findings of exoplanets beyond our solar system.

With the prospect of finding a habitable exoplanet on the horizon, astronomer Schulze-Makuch and his team developed the Earth Similarity Index (ESI) as a tool to assess the similarity of exoplanets to Earth [6]. The ESI became a valuable metric for evaluating the potential habitability of exoplanets by comparing various planetary characteristics with those of Earth. This development opened up new possibilities for understanding and identifying planets that could potentially support life beyond our solar system, marking a significant advancement in the field of astronomy [6].

The implementation of the ESI saw the rise of numerous potential Earth-like planets. One major hurdle that plagued the scientists however was the limits in the data they obtained from telescopes. The transit method, the most popular of the observation methods, relied on the finding of a dip in the brightness of the host star. That dip in brightness and the periodical recurrence of that same dip would indicate the existence of a planet [1].

II. NASA's ESI Variants

Early methods of exo-planet discovery provided a limited amount of data. Most of the available information consisted of the stellar flux and planet radius. The stellar flux, the measure of radiation a planet receives from its host star, became the focal point of the first equation, the Standard ESI equation. Earth's radius (Re) and stellar flux (Fe) were used as the baseline for the equation.

Standard ESI [2]:

$$\mathrm{ESI}(F,R) = 1 - \sqrt{\frac{1}{2}\left[\left(\frac{F-Fe}{F+Fe}\right)^2 + \left(\frac{R-Re}{R+Re}\right)^2\right]}$$

Highest ESI: Teegarden's Star B = 0.9502

The integration of two variables in the Standard ESI yielded Teegarden's Star B as the exo-planet with the greatest similarity to Earth. The planet's orbit in Teegarden's

Star's habitable zone and its very similar radius to Earth reflect the high ESI score. However many in the scientific community still felt the use of two variables was inadequate. As a consequence, a new Weighted ESI equation was formulated to account for more factors.

The Weighted ESI equation utilizes a total of 5 variables: stellar flux, radius, density, escape velocity, and temperature. Each one of these variables has weights that affect the sensitivity of each factor. Temperature holds the highest weight because of the important role it plays in the habitability of planets.

Weighted ESI variables [2]:

Variables	Weights
Flux	1
Radius	0.57
Density	1.07
Escape Velocity	0.70
Temperature	5.58

Weighted ESI:

$$\mathrm{ESI}(F,R,D,E,T) = 1 - \sqrt{\frac{1}{5}\left[\left(\frac{F-Fe}{F+Fe}\right)^2 + (0.57)\left(\frac{R-Re}{R+Re}\right)^2 + (1.07)\left(\frac{D-De}{D+De}\right)^2 + (0.70)\left(\frac{E-Ee}{E+Ee}\right)^2 + (5.58)\left(\frac{T-Te}{T+Te}\right)^2\right]}$$

Highest ESI: Teegarden's Star B = 0.9636

The Weighted ESI's 5 variables deem Teegarden's Star B once again as the planet with the greatest similarity to Earth. The planet's similar radius, mass, and density to Earth suggest that the conditions in that world may be very similar to that of our planet. However Teegarden's Star B does have one startling distinction, the planet orbits a red dwarf star with a mass roughly 9 percent that of our Sun [2].

III. Custom ESI Variants

Red dwarf planetary systems are host to a multitude of issues that may prevent the development of life. The first notable problem lies with the range of the habitable zone. Due to the cooler temperature of red dwarfs, planets in the habitable zone often have orbital periods that range from 5 to 12 days. The close proximity of the planet to its star causes the planet to be tidally locked. One side of the planet would always face the star, and the other side would be left in perpetual night. The absence of a day/night cycle would render most of the surface of such planets unsuitable for life.

Another major problem that plagues red dwarf systems is the radiation of the stars. Red dwarf stars have a life cycle that is considerably longer than that of Sun-sized main sequence stars. Sun-massed stars often live 10 billion years. Red dwarf stars can ensure as long as 1 trillion years. Younger stars tend to send out frequent and more solar storms. Every red dwarf star in the universe is in its infancy. As a consequence, many of the planets orbiting such systems are much more irradiated than Earth or even Mercury [8].

Older ESI models fail to take the mass of the star into account. A planet such as Teegarden's Star B perhaps has a temperature comparable to that of Earth. But it will likely have a surface as irradiated as the ruins of Chornobyl. To mitigate this problem, we have formulated two new variants of the ESI equation. The first of these variants takes in a total of 9 variables, star temperature, and star mass among them, all unweighted. We have titled this first equation the Custom ESI. The structure of the equation is similar to that of the Standard ESI. The one notable difference is the use of 9 variables.

Variables: Flux, Radius, Gravity, Planet Mass, Temperature, Star Temperature, Star Mass, Orbital Period, Density.

Custom ESI:

$$\mathrm{ESI}(F,R,G,M,T,K,S,O,D) = \prod_{i=1}^{9}\left(1-\left|\frac{x_i - x_{io}}{x_i + x_{io}}\right|\right)$$

Highest ESI: Venus = 0.7996

One problem that is immediately apparent with the Custom ESI formula is its inaccuracy. As many in the scientific community are aware, Venus is far too hot to be a habitable planet. The reason for this inaccuracy is due to the failure of the equation to weigh its values. The temperature should hold the highest weight. Factors such as star mass should be included, but they should not allow planets such as Venus to be counted as Earth analogs.

Revised ESI variables [8]:

Variables	Weights
Flux	3.2
Radius	0.57
Gravity	4.75
Planet Mass	0.4
Temperature	10.58
Star Temperature	1
Star Mass	1
Orbital Period	1
Density	2.8

Revised ESI:

$$\mathrm{ESI}(F,R,G,M,T,K,S,O,D) = \prod_{i=1}^{9}\left(1-\left|\frac{x_i - x_{io}}{x_i + x_{io}}\right|\right)^{\frac{wi}{9}}$$

Highest ESI: Kepler-452 B = 0.8478

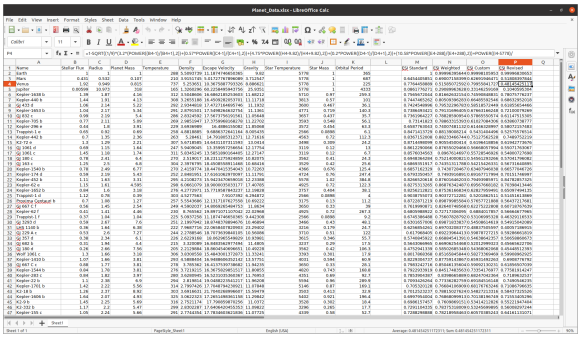
Kepler-452 B orbits a star similar in mass to the Sun. It has a temperature and orbital period similar to that of Earth. The one notable difference is the density of the planet. Earth has a density of 5.51 g/cm3, and Kepler-452 B has a density of 4.21 g/cm3. The exo-planet has a larger mass and radius than Earth. The lower density suggests the planet may be a semi-gaseous planet or an ocean planet [3].

The Revised ESI equation is still not as reliable as it needs to be to find a planet similar to Earth that orbits a star similar to our Sun. The inaccuracies of the 4 ESI variants necessitate the creation of a 5th ESI equation with weights better optimized to search for an Earth analog.

IV. Coding the ESI Variants

The main directory folder of this project includes a Jupyter Notebook file "Astronomy.ipynb" which contains the code and the "Planet_Data.xlsx" file which houses the data. The Excel file contains the data we have gathered from a source that lists every known potentially habitable exo-planet. The code in the Notebook file, which is written in the Julia language, saves the data into a matrix. This matrix is then processed to obtain four Earth Similarity Index (ESI) variants, which are crucial in assessing the similarity of these exoplanets to Earth.

Planet Data Excel Content:



The screenshot above displays the content of the “Planet_Data.xlsx”. The file displays the list of exo-planets and their attributes. The last 4 columns of the files display the results of the 4 ESI equation variants. Earth serves as the baseline point [9].

Defining the Matrix:

```
# Open the Excel file
XLSX.openxlsx("Planet_Data.xlsx", enable_cache=false) do f
    sheet = f["Sheet1"] # changing sheet name leads to error

    # Loop through each row in the sheet
    for r in XLSX.eachrow(sheet)
        # Get the row number
        rn = XLSX.row_number(r)

        # Loop through each column in the row
        for i in 1:num_cols
            # Read the content from the cell and convert to stri
            matrix[rn, i] = string(r[i])
        end
    end

    # Print the matrix with each row on a new line
    println("Matrix:")
    for i in 1:num_rows
        println(join(matrix[i, :], "\t"))
    end
end
```

```
Matrix:
Name    Stellar Flux    Radius  Planet Mass     Temperature
        Star Mass       Orbital Period
Earth   1       1       1       288     5.509373863636364
Mars    0.431   0.532   0.107   210     3.915174516199075
        1       687
Venus   1.92    0.949   0.815   737     5.253650959246789
        1       225
Jupiter 0.00599 10.973  318     165     1.3260296249406163
        1       4333
Kepler-1638 b   1.39    1.87    4.16    312     3.50486064933712
        5710    0.97    259.3
Kepler-440 b    1.44    1.91    4.13    308     3.26551884453919
        3813    0.57    101
GJ 433 d        1.06    2.14    5.22    292     2.93448184759451
```

The “Astronomy.ipynb” processes the data from the “Planet_data.xlsx” file through the use of the “XLSL” package. This Julia package uses the openxlsx() function to temporarily save the data from the input file. Each row of the file object is then read line by line with the eachrow() function. Every element from each row is appended as a string to a matrix titled “matrix”.

The data matrix contains all of the necessary planet data. However, to test out the ESI variant equations, columns 13-16 of the “Planet_data.xlsx” file, the ESI equation results that have already been calculated in the Excel sheet have been excluded from the initial input matrix. The 4 functions: esi_standard, esi_weighted, esi_custom, and esi_revised all replicate each of the 4 ESI variant equations.

ESI Variants Part 1:

```
# defines the functions that represent the 4 original ESI equations

# defines the ESI Standard equation
function esi_standard(f, r)
    # Calculate the intermediate values
    val1 = ((f-1)/(f+1))^2
    val2 = ((r-1)/(r+1))^2

    # Calculate the final result
    result = 1 - sqrt(0.5*(val1 + val2))
    return result
end
```

```
esi_standard (generic function with 1 method)
```

```
# the ESI Weighted equation
function esi_weighted(f, r, d, e, t)
    # Calculate the intermediate values
    val1 = ((f-1)/(f+1))^2
    val2 = 0.57 * ((r-1)/(r+1))^2
    val3 = 1.07 * ((d-5.51)/(d+5.51))^2
    val4 = 0.7 * ((e-11.186)/(e+11.186))^2
    val5 = 5.58 * ((t-288)/(t+288))^2

    # Calculate the final result
    result = 1 - sqrt(0.2*(val1 + val2 + val3 + val4 + val5))
    return result
end
```

ESI Variants Part 2:

```
# the ESI Custom equation
function esi_custom(f, r, d, t, g, m, k, s, o)
    # Calculate the intermediate values
    val1 = ((f-1)/(f+1))^2
    val2 = ((r-1)/(r+1))^2
    val3 = ((d-5.51)/(d+5.51))^2
    val4 = ((t-288)/(t+288))^2
    val5 = ((g-9.82)/(g+9.82))^2
    val6 = ((m-1)/(m+1))^2
    val7 = ((k-5778)/(k+5778))^2
    val8 = ((s-1)/(s+1))^2
    val9 = ((o-365)/(o+365))^2

    # Calculate the final result
    result = 1 - sqrt((1/9)*(val1 + val2 + val3 + val4 + val5 + val6 + val7 + val8 + val9))
    return result
end
```

```
esi_custom (generic function with 1 method)
```

```
# the ESI Revised equation
function esi_revised(f, r, d, t, g, m, k, s, o)
    # Calculate the intermediate values
    val1 = 3.2 * ((f-1)/(f+1))^2
    val2 = 0.57 * ((r-1)/(r+1))^2
    val3 = 2.8 * ((d-5.51)/(d+5.51))^2
    val4 = 10.58 * ((t-288)/(t+288))^2
    val5 = 4.75 * ((g-9.82)/(g+9.82))^2
    val6 = 0.2 * ((m-1)/(m+1))^2
    val7 = ((k-5778)/(k+5778))^2
    val8 = ((s-1)/(s+1))^2
    val9 = ((o-365)/(o+365))^2

    # Calculate the final result
    result = 1 - sqrt((1/9)*(val1 + val2 + val3 + val4 + val5 + val6 + val7 + val8 + val9))
    return result
end
```

```
esi_revised (generic function with 1 method)
```

Aside from the 4 main ESI variants, we also coded another function that represents the 5th ESI variant. We named this new function the ESI Iterative function. This equation inputs 10 variables instead of the 9 of the ESI Custom and ESI Revised equations. By default, the weights of the ESI Iterative equation are set to 1.0. This function be default is only about at accurate as the function that represents the ESI Custom equation. But with some modifications, this equation has the potential to be the most accurate of all.

ESI Iterative Equation:

```
# Sample values for input parameters
f = 1.2
r = 1.1
d = 5.6
e = 11.2
t = 290.0
g = 9.8
m = 1.5
k = 5800.0
s = 1.2
o = 370.0
```

```
370.0
```

```
# uses all 10 variables instead of the 9 of ESI Revised

# the ESI Custom equation
function esi_iterative(f, r, d, e, t, g, m, k, s, o)
    # Calculate the intermediate values
    val1 = w_flux * ((f-1)/(f+1))^2
    val2 = w_radius * ((r-1)/(r+1))^2
    val3 = w_density * ((d-5.51)/(d+5.51))^2
    val4 = w_escape * ((e-11.186)/(e+11.186))^2
    val5 = w_temp * ((t-288)/(t+288))^2
    val6 = w_gravity * ((g-9.82)/(g+9.82))^2
    val7 = w_mass * ((m-1)/(m+1))^2
    val8 = w_stemp * ((k-5778)/(k+5778))^2
    val9 = w_smass * ((s-1)/(s+1))^2
    val10 = w_orbit * ((o-365)/(o+365))^2

    # Calculate the final result
    result = 1 - sqrt((1/10)*(val1 + val2 + val3 + val4 + val5 + val6 + val7 + val8 + val9 + val10))
    return result
end
```

Every variable in the “esi_iterative” function is multiplied by a weight. These weights can be altered through an iterative loop. The screenshot below displays a simple example of a loop of 10 iterations that applies a random value ranging from 0.1 to 19.9 to each of the weight variables. As one can see, this randomized method of assigning weight values tends to yield inaccurate results.

Simple Iterative loop: source – Juliahub.com [14]

```
for i in 1:num_iterations
    # Calculate the current ESI with current weights
    current_esi = calculate_esi(f, r, d, e, t, g, m, k, s, o)
    println("Iteration $i - Current ESI: $current_esi")

    # Update the weights with values ranging from 0.1 to 20
    w_flux += step_size * (0.1 + 19.9 * abs(randn()))
    w_radius += step_size * (0.1 + 19.9 * abs(randn()))
    w_mass += step_size * (0.1 + 19.9 * abs(randn()))
    w_temp += step_size * (0.1 + 19.9 * abs(randn()))
    w_density += step_size * (0.1 + 19.9 * abs(randn()))
    w_escape += step_size * (0.1 + 19.9 * abs(randn()))
    w_gravity += step_size * (0.1 + 19.9 * abs(randn()))
    w_stemp += step_size * (0.1 + 19.9 * abs(randn()))
    w_smass += step_size * (0.1 + 19.9 * abs(randn()))
    w_orbit += step_size * (0.1 + 19.9 * abs(randn()))
end

# Final values of the weights after iterations
println("Final Weights:")
println("w_flux: $w_flux")
println("w_radius: $w_radius")
println("w_mass: $w_mass")
println("w_temp: $w_temp")
println("w_density: $w_density")
println("w_escape: $w_escape")
println("w_gravity: $w_gravity")
println("w_stemp: $w_stemp")
println("w_smass: $w_smass")
println("w_orbit: $w_orbit")
```

```
Iteration 1 - Current ESI: 0.9190848089344853
Iteration 2 - Current ESI: 0.8744242535113019
Iteration 3 - Current ESI: 0.8571416938098567
Iteration 4 - Current ESI: 0.8349806624715916
Iteration 5 - Current ESI: 0.8152098639434567
Iteration 6 - Current ESI: 0.7833755846792504
```

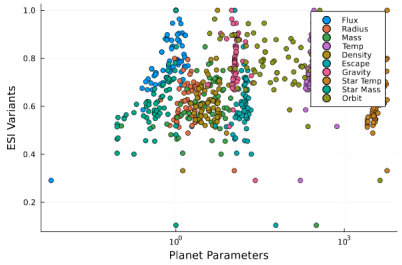
The level of accuracy of these ESI equations can be exemplified by displaying the output of these formulas on a graph. Through the use of the “Plots” package, we can compare the output of the 5 ESI functions. The graph reads the data contained in a newly created “planet_data” matrix. This 2D matrix combines the filtered planet data input matrix, and the matrix that contains the output of the ESI variants.

4 Main ESI Equations Graph:

```
# Extract columns with string data for scatter plot, skipping the first row
x_str = planet_data[2:end, 2:11]  # Assuming string x values are in the second column
y_str = planet_data[2:end, 13:15]  # Assuming string y values are in the third column

# Convert string data to floats
x = parse.(Float64, x_str)  # Convert x_str to Float64 array
y = parse.(Float64, y_str)  # Convert y_str to Float64 array

# Create scatter plot with converted numerical data
scatter(x, y, xlabel="Planet Parameters", ylabel="ESI Variants", legend=:topright, xscale=:log10,
label=["Flux" "Radius" "Mass" "Temp" "Density" "Escape" "Gravity" "Star Temp" "Star Mass" "Orbit"])
```



The Y-axis of this graph represents the ESI score of the planets. The X-axis represents the parameters of the planets. Planets that are more similar to Earth have an ESI value closer to 1. The exo-planets that are more similar to Earth are also clustered toward a few select spots around the middle at the top of the graph.

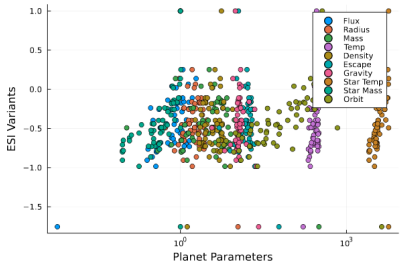
First ESI Iterative Graph:

```
# visualizes the ESI Iterative equation for comparison

x_str = planet_data[2:end, 2:11]  # Assuming string x values are in the second column
y_str = planet_data[2:end, 16]  # Assuming string y values are in the third column

# Convert string data to floats
x = parse.(Float64, x_str)  # Convert x_str to Float64 array
y = parse.(Float64, y_str)  # Convert y_str to Float64 array

# Create scatter plot with converted numerical data
scatter(x, y, xlabel="Planet Parameters", ylabel="ESI Variants", legend=:topright, xscale=:log10,
label=["Flux" "Radius" "Mass" "Temp" "Density" "Escape" "Gravity" "Star Temp" "Star Mass" "Orbit"])
```



The inaccuracy of the first version of the ESI Iterative equation becomes apparent in the graph above. Unlike in the graph of the first 4 ESI formulas, the values of the planet parameters cluster more around the middle of the graph. The tendency for the data to cluster at the center indicates that this early iteration of the ESI Iterative equation is failing to find planets similar to Earth. The only planet that this formula considers Earth-like, is Earth itself.

V. Neural Network Iteration

The 5th ESI equation requires a significant overhaul. While we could have tested out some arbitrary weights again and again until we have finally found a suitable combination of values that could reliably find a planet similar to Earth, we figured the best approach would be to create a neural network to find the answer.

Kepler-442 B Test Case:

```
# loops through the data
for r in 2:64
    # finds the columns of the parameters
    flux = parse(Float64, matrix[r,2])
    radius = parse(Float64, matrix[r,3])
    mass = parse(Float64, matrix[r,4])
    temp = parse(Float64, matrix[r,5])
    density = parse(Float64, matrix[r,6])
    escape = parse(Float64, matrix[r,7])
    gravity = parse(Float64, matrix[r,8])
    stemp = parse(Float64, matrix[r,9])
    smass = parse(Float64, matrix[r,10])
    orbit = parse(Float64, matrix[r,11])

    # filters out by parameters
    if ((flux < 1.3) && (radius < 1.4) && (mass < 3) && (temp < 310) && (
        (escape < 16) && (gravity < 13) && (stemp > 4000) && (smass > 0.5
        print(matrix[r,:],"\t")
    end

    # Kepler-442 b should be the planet most similar to Earth

end
```

```
["Earth", "1", "1", "1", "288", "5.509373863636364", "11.1874746650364
9", "9.82", "5778", "1", "365"] ["Kepler-442 b", "0.7", "1.35", "2.36",
"263", "5.284609995704646", "14.79180531237098", "12.716159122085047", "
4563", "0.72", "112.3"]
```

Before setting up the neural network, the first important step is to find the test case. This test planet should have parameters that are similar to those of Earth. The code above filters every planet with a solar flux less than 1.3 times that of Earth, a radius less than 1.4 times that of Earth, a mass less than 3 times that of our planet, and so forth. A total of 10 parameters were applied to the data until another planet, "Kepler-442 b" remained the only other planet besides Earth in the filtered list.

Neural Network Input: source – Medium.com [15]

```
# Define the loss function
loss(x, y) = Flux.mse(model(x), y)
```

```
loss (generic function with 1 method)
```

```
# Defines the input data of Kepler-442 b
flux = 0.7
radius = 1.35
mass = 2.36
temp = 263
density = 5.284
escape = 14.792
gravity = 12.716
stemp = 4563
smass = 0.72
orbit = 112.3
```

```
112.3
```

```
# Define a function to calculate the loss (e.g., mean squared error)
function loss_function(y_pred, y_true)
    return Flux.mse(y_pred, y_true)
end
```

```
loss_function (generic function with 1 method)
```

```
# Define the target ESI value (ground truth)
# the expected ESI value for Kepler-442 b
target_esi = 0.95

# Define the optimizer (e.g., gradient descent with a learning rate of 0.01)
optimizer = Flux.ADAM(0.01)
```

```
Adam(0.01, (0.9, 0.999), 1.0e-8, IdDict{Any, Any}())
```

```
# defines the selected index to be modified
modify = 1

# defines a list of variable weights
weights = [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]

# define the best weights so far
best_weights = [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
```

The parameters of the exo-planet Kepler-442 B serve as the input parameters for the neural network. The package "Flux" serves as the foundation for the construction of the neural network. The "target_esi" variable is set to a value of 0.95, a value close to 1, but not identical to one since a value of 1 would indicate Earth.

Some similarities between the setup of the first ESI iterative equation and the neural network setup can be seen in the screenshot above and the screenshot below. This neural network includes a new revised version of the ESI Iterative equation. While the old version of this function used individual weight values each created as a lone float value. This new ESI Iterative function utilizes a list of weights. The use of the weight list serves several important tasks in the neural network.

New ESI Iterative [15]:

```
# a revised version of the iterative equation, uses the list of weights instead of the weight varaibles

function esi_iterative(f, r, d, e, t, g, m, k, s, o)
    # Calculate the intermediate values
    val1 = abs(weights[1] * ((f-1)/(f+1))^2)
    val2 = abs(weights[2] * ((r-1)/(r+1))^2)
    val3 = abs(weights[7] * ((m-1)/(m+1))^2)
    val4 = abs(weights[5] * ((t-288)/(t+288))^2)
    val5 = abs(weights[3] * ((d-5.51)/(d+5.51))^2)
    val6 = abs(weights[4] * ((e-11.186)/(e+11.186))^2)
    val7 = abs(weights[6] * ((g-9.82)/(g+9.82))^2)
    val8 = abs(weights[8] * ((k-5778)/(k+5778))^2)
    val9 = abs(weights[9] * ((s-1)/(s+1))^2)
    val10 = abs(weights[10] * ((o-365)/(o+365))^2)

    # Calculate the final result
    result = 1 - sqrt((1/10)*(val1 + val2 + val3 + val4 + val5 + val6 + val7 + val8 + val9 + val10))
    return result
end
```

```
esi_iterative (generic function with 1 method)
```

```
prev_loss = Inf # Initialize previous loss with a large value or any suitable initial value

# Define the chain dense model
model = Chain(
    Dense(10, 16, relu),
    Dense(16, 8, relu),
    Dense(8, 1)
)

# Define the loss function
loss_function(y_pred, y_target) = Flux.mse(y_pred, y_target)

# defines best esi
best_esi = 0

# Define the training loop
prev_loss = Inf
for i in 1:1000
    # Compute the ESI prediction
    esi_pred = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass, orbit)

    # Compute the loss between the prediction and the target
    curr_loss = loss_function(esi_pred, target_esi)

    # Compute the scalar loss value as the sum of elements in curr_loss
    scalar_loss = sum(curr_loss)

    # Compute the gradient of the loss with respect to the model parameters
    grad = gradient(() -> scalar_loss, Flux.params(model))
```

The "weights" list stores a list of floats that are set to an initial value of 1.0. A series of if else statements examine if the current ESI value is less than or greater than the previous ESI score. A variable titled "modify" determines what index of the "weights" list to increment or decrement.

Neural Network Iteration [15]:

```
    # Compute the gradient of the loss with respect to the model parameters
    grad = gradient(() -> scalar_loss, Flux.params(model))

    # Update the weights using the gradient information, if gradient is not nothing
    learning_rate = 0.001  # Define the learning rate
    for (param, gradient) in zip(Flux.params(model), grad)
        if gradient !== nothing
            # Update the weight by subtracting the gradient multiplied by the learning rate
            Flux.update!(param, -learning_rate .* gradient)
        end
    end

    # Extract the new ESI value
    new_esi = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass, orbit)
    #print(new_esi)
    # saves best weight only if ESI is the best ESI
    if new_esi >= best_esi
        best_esi = new_esi
        print(best_esi)
        best_weights = copy(weights)
    end

    # Compare current loss to previous loss
    if curr_loss < prev_loss
        println("Loss decreased!")
        # Increase the value of the selected index
        weights[modify] = min(weights[modify] + 0.1 * i, 20)

    else
        println("Loss stayed the same or increased!")
        # Decrease the value of the selected index using exponential decay
        if modify == 10
            # If modify is 10, use subtraction
            weights[modify] = max(weights[modify] - 0.1 * i, 0.1)
        else
            # If modify is not 10, use absolute value to determine whether to add or subtract
            if rand() < 0.5
                weights[modify] = min(weights[modify] + 0.1 * i, 20)
            else
                weights[modify] = max(weights[modify] - 0.1 * i, 0.1)
            end
        end

        # Move to the next index
        modify = mod(modify + 1, 10)

        # Reset modify value to 1 if it becomes 0 after modulo operation
        if modify == 0
            modify = 10
        end
    end

    # Print the updated weights and ESI value
    println("Epoch: $i, Current ESI: $esi_pred, Current Loss: $curr_loss, Index Modified: $modify")

    prev_loss = curr_loss # Update previous loss with current loss for next iteration
end
```

Another variable called "best_esi" stores the highest calculated ESI value throughout the for loop. The accompanying variable "best_weights" stores the value of the "weights" list when a new "best_esi" value is found. The purpose of the neural network is to find the combination of weight increments and decrements that results in the highest possible ESI values.

Highest ESI Value:

```
planet_data
```

```
64×16 Matrix{String}:
 "Name"              "Stellar Flux"  "Radius"  …  "ESI Iterative"
 "Earth"             "1"             "1"          "0.9999912977166638"
 "Mars"              "0.431"         "0.532"      "0.8817781499455846"
 "Venus"             "1.92"          "0.949"      "0.9397477594365703"
 "Jupiter"           "0.00599"       "10.973"     "0.7880079503025605"
 "Kepler-1638 b"     "1.39"          "1.87"    …  "0.9212596458160197"
 "Kepler-440 b"      "1.44"          "1.91"       "0.8976679183400292"
 "GJ 433 d"          "1.06"          "2.14"       "0.8736370739579886"
 "Kepler-1653 b"     "1.04"          "2.17"       "0.9004938133574363"
 "GJ 832 c"          "0.99"          "2.19"       "0.8716631106917544"
 "Kepler-705 b"      "0.77"          "2.11"    …  "0.8810896337355588"
 "Kepler-296 e"      "0.44"          "1.8"        "0.8827626443015071"
 "Trappist-1 e"      "0.65"          "0.92"       "0.8627545554910138"
 ⋮                                               ⋮
 "TOI-700 d"         "0.87"          "1.14"       "0.9015515042601017"
 "Kepler-1649 c"     "1.23"          "1.06"       "0.8835974350942897"
 "HD 40307 g"        "0.67"          "2.57"       "0.8929552244596236"
 "Kepler-296 f"      "0.44"          "1.8"     …  "0.8827626443015071"
 "HIP 38594 b"       "1.34"          "2.77"       "0.868771064110056"
 "K2-288 B b"        "0.44"          "1.91"       "0.866559021252137l"
 "HD 216520 c"       "1.28"          "3.04"       "0.8804445234448385"
 "Teegarden's Star c"  "0.37"        "1.04"       "0.8606950370430563"
 "Kepler-1229 b"     "0.32"          "1.4"     …  "0.8955440832713132"
 "Kepler-186 f"      "0.29"          "1.17"       "0.9112692128962312"
 "GJ 667 C e"        "0.56"          "1.45"       "0.885971555218888"
 "Trappist-1 g"      "0.25"          "1.13"       "0.8539073526693943"
```

```
# gets the data of Kepler-442 b
for r in 2:64
    if planet_data[r,1] == "Kepler-442 b"
        print(planet_data[r,:])
    end
end
```

```
["Kepler-442 b", "0.7", "1.35", "2.36", "263", "5.284609995704646", "14.79180531237098", "1
2.716159122085047", "4563", "0.72", "112.3", "0.8367152007844013", "0.8823346674351574", "0.
7512756252815086", "0.7489752218963184", "0.9241026555452485"]
```

After the for loop of 1000 iterations has been completed, the data from the "best_weights" list is processed by the new ESI Iterative equation. This iteration formula is then used to update the calculations found in column 16 of the "planet_data" matrix. The ESI value for Kepler-442 b after this iteration of the neural network is equal to about 0.924, a value close to 1.

Best Weights Results:

```
: best_weights
```

```
: 10-element Vector{Float64}:
   0.1
   0.1
  20.0
  20.0
  20.0
  20.0
   0.1
   0.1
  20.0
   0.1
```

The values in the "best_weights" list depicted above indicate that values such as planet mass, temperature, escape velocity, gravity, and star mass should have a weight of 20. The rest of the weights are set to 0.1 which is almost akin to not having those values present at all. The list of "best_weights" values present in the screenshot above is only one example of possible values that can lead to a high ESI.

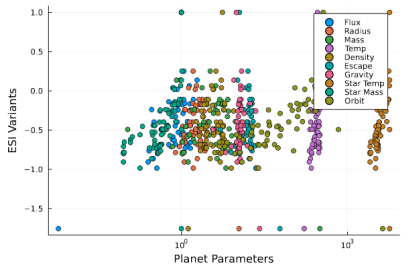
Old ESI Iterative Graph:

```
# visualizes the ESI Iterative equation for comparison

x_str = planet_data[2:end, 2:11]  # Assuming string x values are in the second column
y_str = planet_data[2:end, 16]  # Assuming string y values are in the third column

# Convert string data to floats
x = parse.(Float64, x_str)  # Convert x_str to Float64 array
y = parse.(Float64, y_str)  # Convert y_str to Float64 array

# Create scatter plot with converted numerical data
scatter(x, y, xlabel="Planet Parameters", ylabel="ESI Variants", legend=:topright, xscale=:log10,
label=["Flux" "Radius" "Mass" "Temp" "Density" "Escape" "Gravity" "Star Temp" "Star Mass" "Orbit"])
```



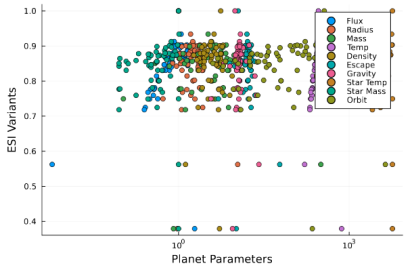
Final ESI Iterative Graph:

```
# visualizes the ESI Iterative equation for comparison

x_str = planet_data[2:end, 2:11]  # Assuming string x values are in the second column
y_str = planet_data[2:end, 16]  # Assuming string y values are in the third column

# Convert string data to floats
x = parse.(Float64, x_str)  # Convert x_str to Float64 array
y = parse.(Float64, y_str)  # Convert y_str to Float64 array

# Create scatter plot with converted numerical data
scatter(x, y, xlabel="Planet Parameters", ylabel="ESI Variants", legend=:topright, xscale=:log10,
label=["Flux" "Radius" "Mass" "Temp" "Density" "Escape" "Gravity" "Star Temp" "Star Mass" "Orbit"])
```



One notable difference between the first ESI Iterative function and the final neural network-optimized ESI Iterative function lies in the distribution of the ESI values. The distribution of the old version saw a clustering of values in the center of the graph, suggesting a lack of accuracy. The distribution of the second graph shows a higher concentration of ESI values toward the upper central area of the graph, indicating an improvement in the reliability of the function.

If we examine the composition of the planet Kepler-442 B, the planet that the final ESI Iterative equation deemed the most Earth-like, we can find a number of similarities between the exo-planet and our home planet. The mass of Kepler-442 B is 2.36 times that of Earth. The temperature is 263 degrees Kelvin. The escape velocity is 14.79 km/sec, and the gravity is 12.716 m/s^2. The planet's star is 0.72 times that of our Sun. The planet's density is also similar to Earth at 5.285 g/cm^3.

Kepler-442 B's similarity to Earth could be a result of a reliable neural network, but it could also be a result of simple fortune. The list of weights does tend to vary significantly between each run of the iterations. This instability in the network is only one on the list of issues that we had to overcome, and still need to correct in some cases.

VI. Challenges in Coding

In this iteration of our program, we did not encounter any problems with the installation of packages. However, we did encounter various challenges throughout the setup of the code itself. The first set of errors were simple syntax errors that were corrected by finding an alternative and more reliable method of doing a particular algorithm.

One example of such a simple error lies in the initial setup of the input matrix. The "sheet_size()" function of the "XLSX" was throwing out an error. As an alternative, we opted to manually declare the size of the dimensions of the matrix.

Another example of a relatively simple error we encountered was the "domain error", which was the result of attempting to create the ESI equations in just one line. Separating the equations into values that could then be square-rooted simplified the process of debugging the code and finding the source of the domain error.

Domain Error:

```
# Uses the columns from the first matrix to calculate the values of the ESI matrix
flux = 0.0
radius = 0.0
mass = 0.0
temp = 0.0
density = 0.0
escape = 0.0
gravity = 0.0
stemp = 0.0
smass = 0.0
orbit = 0.0

# Loops from the second row of the first matrix to the end
for r in 2:64

    # Stores the values of the ESI variables
    flux = parse(Float64, matrix[r,2])
    radius = parse(Float64, matrix[r,3])
    mass = matrix[r,4]
    temp = matrix[r,5]
    density = matrix[r,6]
    escape = matrix[r,7]
    gravity = matrix[r,8]
    stemp = matrix[r,9]
    smass = matrix[r,10]
    orbit = matrix[r,11]

    # Fills in values of the ESI Standard equation
    esi[r-1,1] = s(flux, radius)
    #print()
end
```

```
DomainError with -0.3054830287206266:
sqrt will only return a complex result if called with a complex argument. Try sqrt(Complex(x)).
```

Domain Error Fixed:

```
# defines the functions that represent the 4 original ESI equations

# defines the ESI Standard equation
function esi_standard(f, r)
    # Calculate the intermediate values
    val1 = ((f-1)/(f+1))^2
    val2 = ((r-1)/(r+1))^2

    # Calculate the final result
    result = 1 - sqrt(0.5*(val1 + val2))
    return result
end
```

```
esi_standard (generic function with 1 method)
```

```
# the ESI Weighted equation
function esi_weighted(f, r, d, e, t)
    # Calculate the intermediate values
    val1 = ((f-1)/(f+1))^2
    val2 = 0.57 * ((r-1)/(r+1))^2
    val3 = 1.07 * ((d-5.51)/(d+5.51))^2
    val4 = 0.7 * ((e-11.186)/(e+11.186))^2
    val5 = 5.58 * ((t-288)/(t+288))^2

    # Calculate the final result
    result = 1 - sqrt(0.2*(val1 + val2 + val3 + val4 + val5))
    return result
end
```

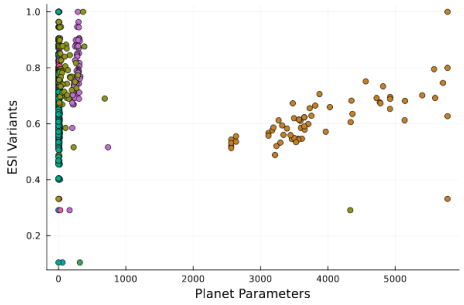
The last example of a simple error we encountered had to do with the display of data in graphs. The first graph we created from the "planet_data" matrix used a numerical scale system to display the data points. If we intended to display every planet attribute one by one, the numerical scale should have been sufficient. However, because we wanted to display all data points in one graph, we had to find a method to display the data in a visually accurate manner. The question for us was, how do we display the Earth masses which ranged from 1 to 318, and star temperatures which ranged from 2600 to 5800 in one graph?

Numerical Scale:

```
# Extract columns with string data for scatter plot, skipping the first row
x_str = planet_data[2:end, 2:11]  # Assuming string x values are in the second column
y_str = planet_data[2:end, 13:15]  # Assuming string y values are in the third column

# Convert string data to floats
x = parse.(Float64, x_str)  # Convert x_str to Float64 array
y = parse.(Float64, y_str)  # Convert y_str to Float64 array

# Create scatter plot with converted numerical data
scatter(x, y, xlabel="Planet Parameters", ylabel="ESI Variants", legend=false)
```



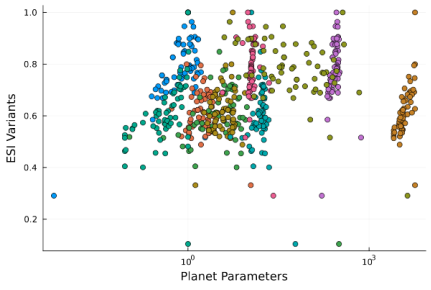
As one can notice, the vast differences in scale pose a problem if we want to create a visual sense of how data of varying scales vary. Fortunately, a simple solution to this issue exists. To display the star temperatures and Earth masses in a visually scalable manner, all we had to do was set the xscale to "log10". With this one change, we could now more easily see the connection between the planet attributes and the ESI scores.

Log10 Scale:

```
# Extract columns with string data for scatter plot, skipping the first row
x_str = planet_data[2:end, 2:11]  # Assuming string x values are in the second column
y_str = planet_data[2:end, 13:15]  # Assuming string y values are in the third column

# Convert string data to floats
x = parse.(Float64, x_str)  # Convert x_str to Float64 array
y = parse.(Float64, y_str)  # Convert y_str to Float64 array

# Create scatter plot with converted numerical data
scatter(x, y, xlabel="Planet Parameters", ylabel="ESI Variants", legend=false, xscale=:log10)
```



The challenges we encountered previously fell into the category of simple issues with simple solutions. The next set of challenges we encountered however required more throughout debugging and re-planning of the code in some instances. We encountered a notable number of these challenges in the creation of the neural network.

Random Indexes:

```
# Define the loss function
loss_function(y_pred, y_target) = Flux.mse(y_pred, y_target)

# Define the training loop
prev_loss = Inf
for i in 1:100
    # Compute the ESI prediction
    esi_pred = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass, orbit)

    # Compute the loss between the prediction and the target
    curr_loss = loss_function(esi_pred, target_esi)

    # Compute the scalar loss value as the sum of elements in curr_loss
    scalar_loss = sum(curr_loss)

    # Compute the gradient of the loss with respect to the model parameters
    grad = gradient(() -> scalar_loss, Flux.params(model))

    # Update the weights using the gradient information, if gradient is not nothing
    learning_rate = 0.01  # Define the learning rate
    for (param, gradient) in zip(Flux.params(model), grad)
        if gradient !== nothing
            # Update the weight by subtracting the gradient multiplied by the learning rate
            Flux.update!(param, -learning_rate .* gradient)
        end
    end

    # Extract the new ESI value
    new_esi = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass, orbit)

    # Compare current loss to previous loss
    if curr_loss < prev_loss
        println("Loss decreased!")
        # increases the value of a random index
        rn = rand(1:10)
        weights[rn] = weights[rn] + 0.1 * i
```

The first version of the neural network utilized a random number generator to determine what index to update. Needless to say this approach was not efficient. To improve the efficiency of the algorithm, we created a "best_weights" list to store the weights of the highest ESI result.

Best Weights List:

```
# defines the selected index to be modified
modify = 1

# defines a list of variable weights
weights = [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]

# define the best weights so far
best_weights = [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0]
```

```
10-element Vector{Float64}:
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
 1.0
```

As we made our first attempt to use this list, we soon noticed that the output was not producing the output we expected. The first unexpected result we got was a list of negative weights. We set a range of 0.1 to 20. And as soon as we made those corrections, we found another issue with the storage of the "best_weights" list. Instead of getting the weights of the instance of the highest ESI, we were getting the weights of the most recent instance in which the loss decreased.

Weights Range Set:

```
# Extract the new ESI value
new_esi = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass, orbit)

# Compare current loss to previous loss
if curr_loss < prev_loss
    println("Loss decreased!")
    # Increase the value of the selected index
    weights[modify] = min(weights[modify] + 0.1 * i, 20)
else
    println("Loss stayed the same or increased!")
    # Decrease the value of the selected index using exponential decay
    if modify == 10
        # If modify is 10, use subtraction
        weights[modify] = max(weights[modify] - 0.1 * i, 0.1)

        # saves the best weights if loss is decreasing
        best_weights = weights
    else
        # If modify is not 10, use absolute value to determine whether to add or subtract
        if rand() < 0.5
            weights[modify] = min(weights[modify] + 0.1 * i, 20)
        else
            weights[modify] = max(weights[modify] - 0.1 * i, 0.1)
        end
    end

    # Move to the next index
    modify = mod(modify + 1, 10)

    # Reset modify value to 1 if it becomes 0 after modulo operation
    if modify == 0
        modify = 10
    end
end
```

We added another variable titled "best_esi in hopes of mitigating this issue. We continued to obtain the incorrect weight values from the "best_weights" list. We went through a number of different versions of the code below until we concluded that we needed to create a copy of "weights".

Best ESI Copy:

```
# defines best esi
best_esi = 0

# Define the training loop
prev_loss = Inf
for i in 1:1000
    # Compute the ESI prediction
    esi_pred = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass

    # Compute the loss between the prediction and the target
    curr_loss = loss_function(esi_pred, target_esi)

    # Compute the scalar loss value as the sum of elements in curr_loss
    scalar_loss = sum(curr_loss)

    # Compute the gradient of the loss with respect to the model parameters
    grad = gradient(() -> scalar_loss, Flux.params(model))

    # Update the weights using the gradient information, if gradient is not nothing
    learning_rate = 0.001  # Define the learning rate
    for (param, gradient) in zip(Flux.params(model), grad)
        if gradient !== nothing
            # Update the weight by subtracting the gradient multiplied by the learning rate
            Flux.update!(param, -learning_rate .* gradient)
        end
    end

    # Extract the new ESI value
    new_esi = esi_iterative(flux, radius, density, escape, temp, gravity, mass, stemp, smass,
    #print(new_esi)
    # saves best weight only if ESI is the best ESI
    if new_esi >= best_esi
        best_esi = new_esi
        print(best_esi)
        best_weights = copy(weights)
    end

    # Compare current loss to previous loss
    if curr_loss < prev_loss
        println("Loss decreased!")
        # Increase the value of the selected index
        weights[modify] = min(weights[modify] + 0.1 * i, 20)
```

This declaration of a copy while updating the "best_weights" list insured that the data we were obtaining was the unmodified version of the weight data at the moment the highest ESI was obtained.

VII. Conclusion

In conclusion, our ESI astronomy project has made significant progress in researching and developing a reliable method to find habitable, Earth-like planets beyond our solar system. We have explored various mathematical aspects of the project, including the development of a Revised ESI formula that takes into account new factors affecting planetary habitability. We have also successfully implemented iterative methods, utilizing neural networks with weights and biases, to further refine the accuracy of the ESI formula.

Additionally, we have also created a fifth variant of the ESI formula, called the ESI Iterative equation, which incorporates iterative methods and neural networks to estimate the ideal weights for finding exoplanets most similar to Earth. While the neural network may have its limitations, it can potentially uncover valuable clues about which parameters to emphasize in determining planetary habitability.

With the progress made so far and the incorporation of numerical computing techniques and elements from artificial intelligence, particularly deep learning, we are optimistic about the future stages of our project and the potential implications for the field of astronomy. We will continue to translate relevant code into Julia and diligently work towards achieving our project goals.

References

[1] P. Brennan, "Exoplanet Catalog", *Exoplanets.nasa.gov*, p. 1-3, 2022.

[2] A. Mendez, "Earth Similarity Index (ESI)", *PHL.upr.edu*, 2-4, 2023.

[3] J. Papiewski, "How to Measure the Density of a Planet", *Sciencing.com*, p. 2-3, 2017.

[4] B. Szyk, "Escape Velocity Calculator", *Omnicalculator.com*, p. 1-6, 2023.

[5] A. Anton, "Acceleration of Gravity", *Planetcalc.com*, p. 1-2, 2021.

[6] J. Trosper, "Earth-Similarity Index: Where Could We Live Besides Earth?", *Interestingengineering.com*, p. 3-7, 2021.

[7] O. Markus, "Math Equation Editor", *Imatheq.com*, p. 1, 2021.

[8] B. Hays, "NASA: Red dwarf habitable zones may not be so habitable", *UPI.com*, p. 1-4, 2017.

[9] A. Mendez, "The Habitable Exoplanets Catalog", *PHL.upr.edu*, 2-5, 2023.

[10] B. Andriamanalimana, "Software part 2 Julia", *Module 1*, p. 35-55, 2023.

[11] B. Damya, "Iteratively Fine-Tuning Neural Networks with Weights & Biases", *Wandb.ai*, p. 1-7, 2019.

[12] M. Ridul, "Working with Excel Files in Julia", *Geeksforgeeks.org*, p. 1-3, 2020.

[13] B. Andriamanalimana, "Class 2 Elementary Julia", *Module 3*, 2023.

[14] "Iterative Solvers as Iterators", JuliaHub.com, p. 1-3, 2023.

[15] D. Boudreau, "Deep Learning with Julia", Medium.com, p. 1-7, 2021.