Proyecto de Investigación

Cristhiam Daniel Campos Julca

26 de agosto de 2022

1. Simulación en Matlab

En primer lugar se implementar en Simulink el modelo de un arreglo fotovoltaico, con paneles del modelo Sunset PX 72, que cuenta con 72 celdas de silicio polycristalino. Con la finalidad de obtener una potencia aproximada de 4.073 kW se colocan cuatro paneles en serie y tres en paralelo.

En la siguiente tabla se pueden observar los datos proporcionados por el panel Sunset PX a condiciones estándar de prueba, bajo una temperatura de 298.15 K equivalente a 25° C y una radiación de 1000 W/m2.

Datos bajo condiciones estándar	**STD**
Potencia en el punto máximo (P_{max})	340 W
Tensión en circuito abierto (V_{oc}])	47.4 V
Tension en el punto de maxima potencia (V_{mpp})	38.4 V
Corriente de cortocircuito (I_{sc})	9.35 A
Corriente en el punto de máxima potencia (I_{mpp})	8.84 A
Numero de celdas (N_s)	72
Coeficiente de Temperatura (I_{sc})	0.037 % /K
Coeficiente de Temperatura (V_{oc})	-0.32 % /K
Resistencia en serie (R_s)	0.39 Ω
Resistencia en paralelo (R_{sh})	545.82 Ω

Para esta primera simulación se considera las entradas constantes, es decir una temperatura de 25°C y una irradiancia de 1000 W/m^2

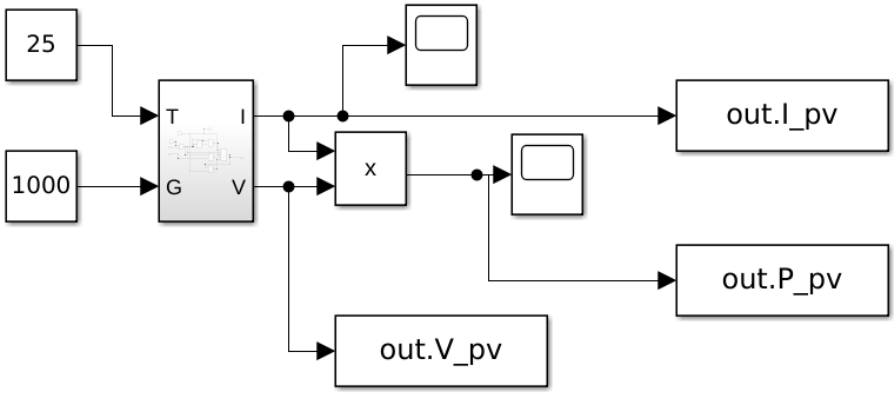

Figura 1: Arreglo PV

Así mismo, se parte del modelo de los cinco parámetros, en donde la corriente y la tensión de salida se ven gobernadas por los siguientes ecuaciones representadas en subsistemas:

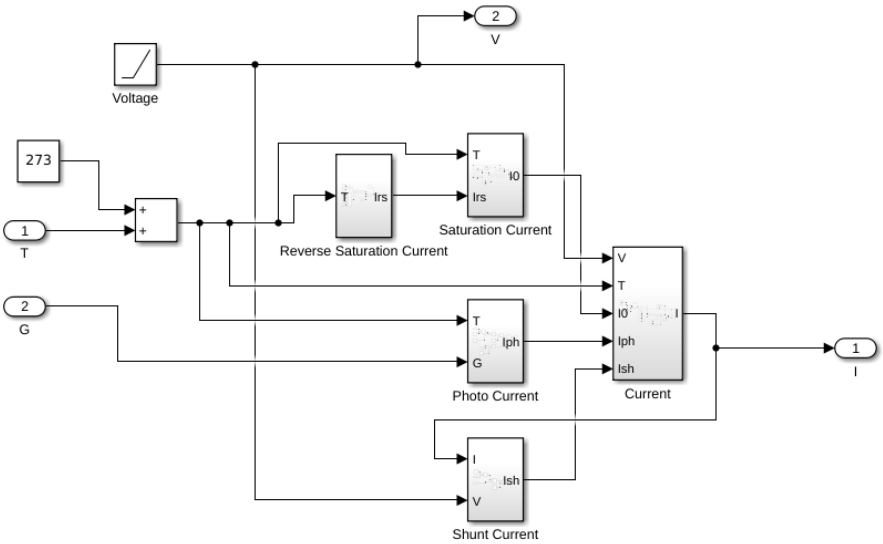

Figura 2: Sub-sistemas del modelo

- Fotocorriente

$$I_{ph} = (I_{sc} + K_i \times (T - 298)) \times (G/1000)$$

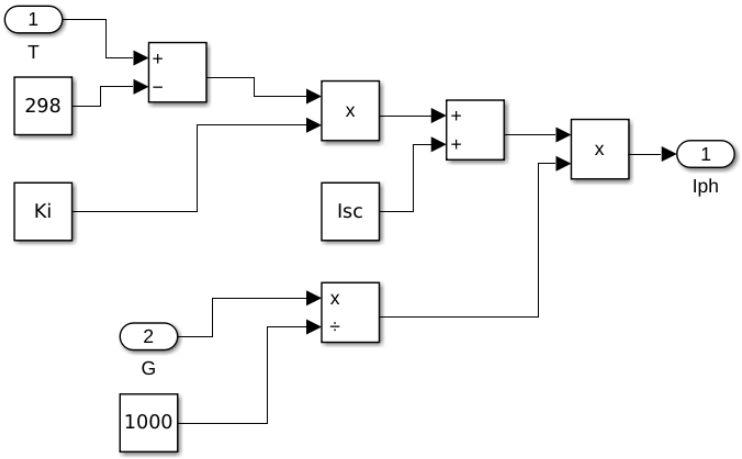

Figura 3: Subsistema para Fotocorriente

- Corriente de saturación

$$I_o = I_{rs} \times \left(\frac{T}{T_n}\right)^3 \times \exp\left((q \times E_{go} \times (1/T_n - 1/T)) / (n \times k)\right)$$

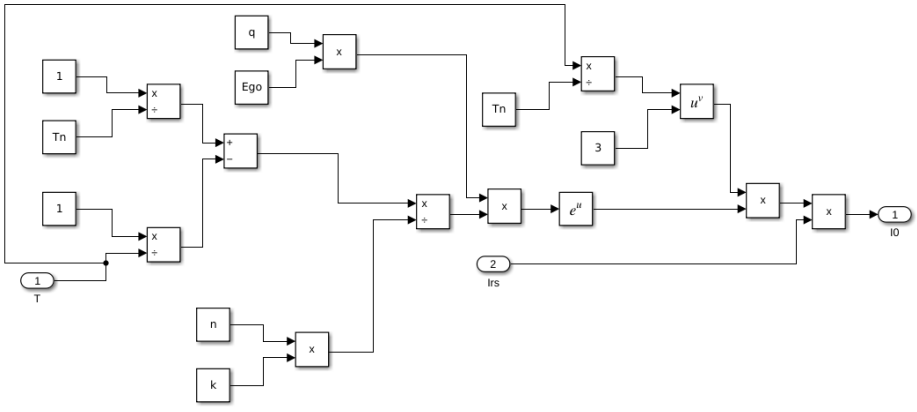

Figura 4: Subsistema de Corriente de saturación

- Corriente de saturación reversa

$$I_{rs} = I_{sc}/(\exp((q * V_{oc})/(n * N_s * k * T)) - 1)$$

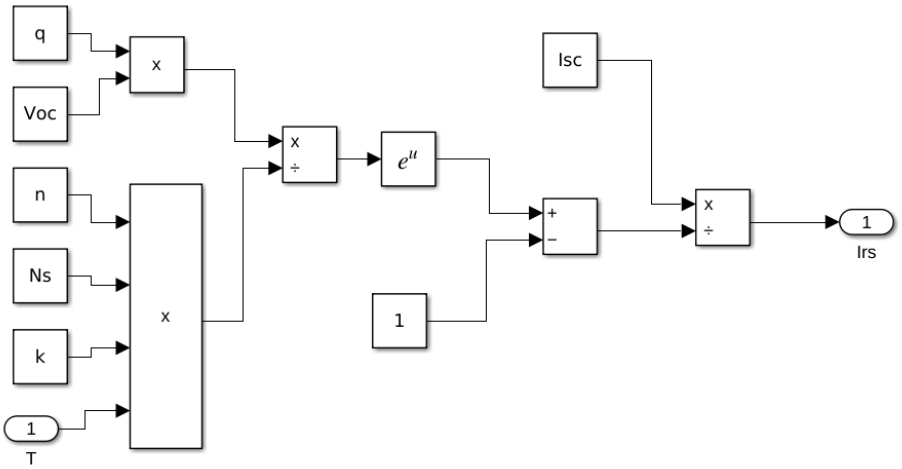

Figura 5: Subsistema de Corriente de saturación reversa

- Corriente shunt

$$I_{sh} = (V + I * R_s)/R_{sh}$$

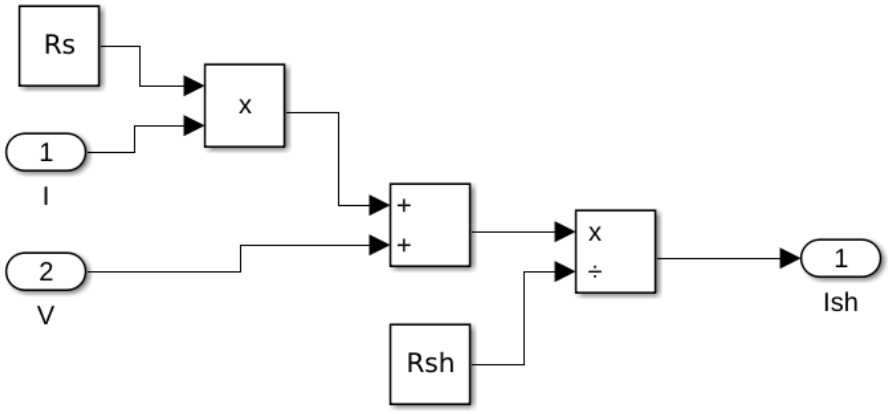

Figura 6: Subsistema de la corriente shunt

- Corriente de salida

$$I = I_{ph} * NP - I_o * NP * (\exp((q * (V/NS + I * R_s/NP))/(n * N_s * k * T)) - 1) - I_{sh}$$

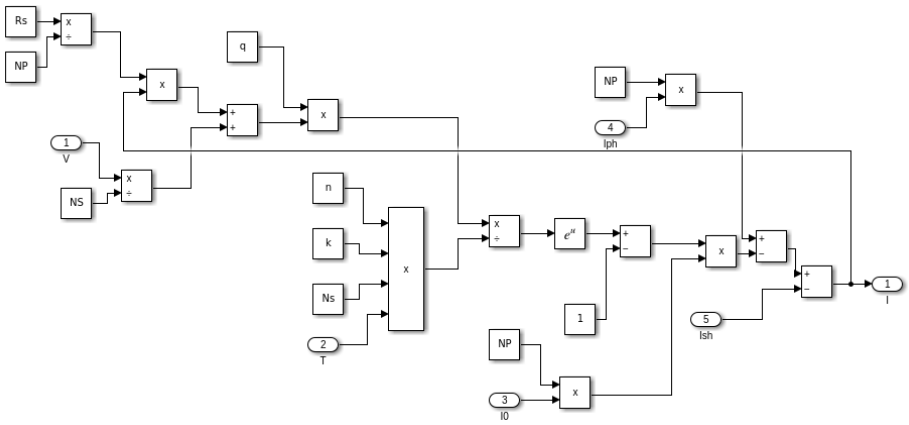

Figura 7: Subsistema de la corriente de salida

Los resultados son almacenados en un archivo CSV para su fácil manipulación en otros lenguajes como Python.

Las curvas características que se obtienen del arreglo fotovoltaico son:

```
def curvaCaracteristica():
    v = df.V_pv
    i = df.I_pv
    p = df.P_pv

    fig, ax1 = plt.subplots()

    color = "tab:red"
    ax1.set_xlabel("Tension (V)")
    ax1.set_ylabel("Corriente (A)", color=color)
    ax1.plot(v, i, color=color)
    ax1.tick_params(axis="y", labelcolor=color)
    ax1.set_xlim(0,195)
    ax1.set_ylim(0,30)
    ax2 = ax1.twinx()

    color = "tab:blue"
    ax2.set_ylabel("Potencia (W)", color=color)
    ax2.plot(v, p, color=color)
    ax2.tick_params(axis="y", labelcolor=color)
    ax2.set_xlim(0,195)
```

```
    ax2.set_ylim(0,4000)

    fig.tight_layout()
    plt.grid()

    plt.savefig("./imagenes/curvacaracteristica.png")
    plt.show()
```

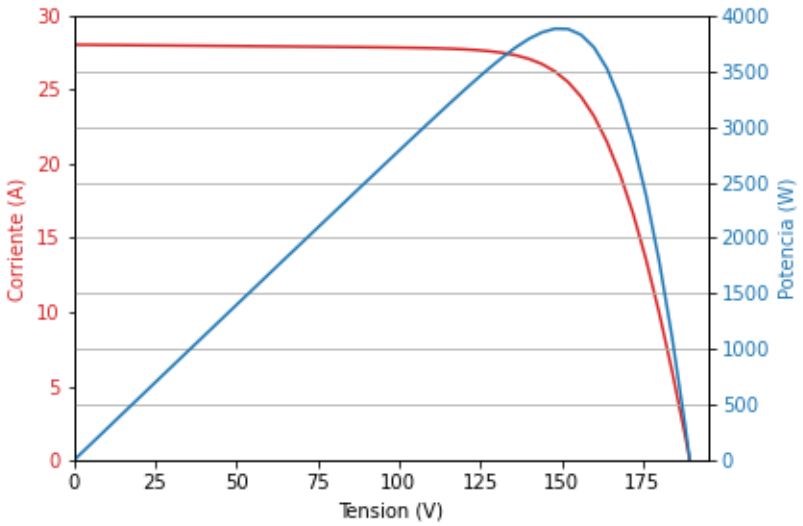

Figura 8: Curvas características

Se hace uso de la función `numpy.poly1d()` que nos ayuda a definir una función polinomial de la siguiente manera:

```
def curvasAjuste():
    #fit polynomial models up to degree 5
    model1 = np.poly1d(np.polyfit(df.V_pv, df.I_pv, 1))
    model2 = np.poly1d(np.polyfit(df.V_pv, df.I_pv, 2))
    model3 = np.poly1d(np.polyfit(df.V_pv, df.I_pv, 3))
    model4 = np.poly1d(np.polyfit(df.V_pv, df.I_pv, 4))
    model5 = np.poly1d(np.polyfit(df.V_pv, df.I_pv, 5))

    #create scatterplot
    polyline = np.linspace(1, 200, 100)
    plt.scatter(df.V_pv, df.I_pv)

    #add fitted polynomial lines to scatterplot
    plt.plot(polyline, model1(polyline), color='green',
   label= 'Funcion grado 1')
    plt.plot(polyline, model2(polyline), color='red',
   label= 'Funcion grado 2')
```

```
    plt.plot(polyline, model3(polyline), color='purple',
    label= 'Funcion grado 3')
    plt.plot(polyline, model4(polyline), color='blue',
    label= 'Funcion grado 4')
    plt.plot(polyline, model5(polyline), color='orange',
    label= 'Funcion grado 5')

    plt.legend()
    plt.grid()
    plt.savefig("./imagenes/curvaajuste.png")
    plt.show()
```

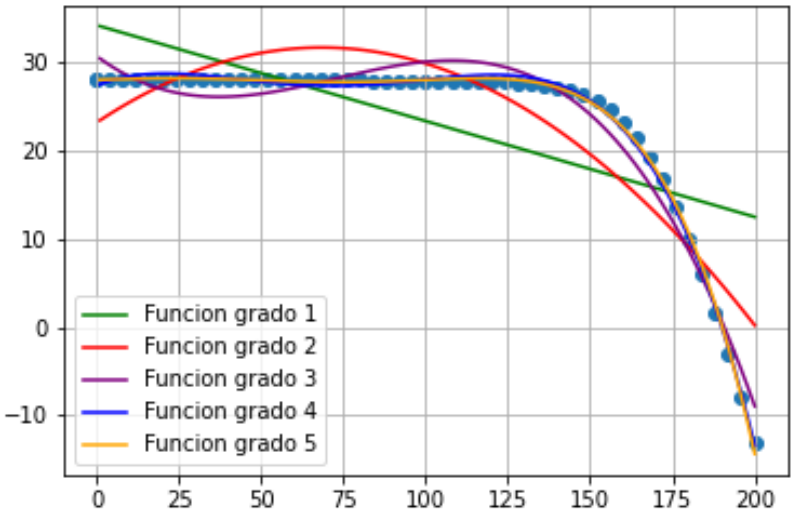

Figura 9: Funciones Polinomiales

Los resultados del R ajustado para cada modelo se obtienen de la siguiente función, y están organizados en la tabla:

```
def adjR(x, y, degree):
    results = {}
    coeffs = np.polyfit(x, y, degree)
    p = np.poly1d(coeffs)
    yhat = p(x)
    ybar = np.sum(y)/len(y)
    ssreg = np.sum((yhat-ybar)**2)
    sstot = np.sum((y - ybar)**2)
    results['r_squared'] = 1- (((1-(ssreg/sstot))*(len(y)
    -1))/(len(y)-degree-1))

```

```
11      return results
```

Modelo	R cuadrado ajustado
Grado 1	0.4376094058571091
Grado 2	0.7969866148484753
Grado 3	0.9606980241040581
Grado 4	0.9968071975795516
Grado 5	0.9984087544931406

Se selecciona el modelo de grado 5:

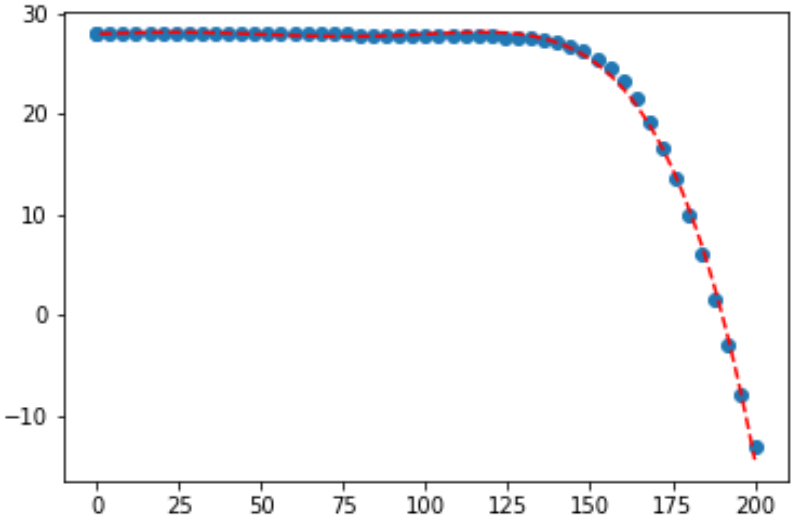

Figura 10: Modelo seleccionado

El resultado de los coeficientes de la función polinómica de grado 5 es:

$$i(v) = -8{,}713\times10^{-10}v^5+2{,}213\times10^{-7}v^4-1{,}538\times10^{-5}v^3+7{,}92\times10^{-5}v^2+0{,}01166v+27{,}93$$