```
> restart;
```

PV generator model

In this MAPLE worksheet, the Jacobian matrix is applied to the PV generator model to calculate the first ten iterations of the open-circuit voltage $U_{OC}(\vartheta_C, \Phi_G)$ in (V) and the reverse saturation current $I_S(\vartheta_C)$. The calculations were performed for different values of the PV cell temperature ϑ_C and the irradiance E_G. However, in this printout ϑ_C= 25°C and E_G = 200 Wm^{-1}.

Header

Used mathematical packages:

```
> with(LinearAlgebra):
> with(VectorCalculus, Jacobian):
```

Parameters

Specifications of the PV generator:

```
> param := [ I__SC_STC = 9.79, U__OC_STC = 24.27, m = 1, N_C = 36,
  E__STC = 1000, E__G = 200, k = 1.380649 * 10^(-23), e =
  1.602176634 * 10^(-19), vartheta__C = 25, vartheta__STC = 25,
  TC__I_SC = 0.05, TC__U_OC = -0.29 ];
```

$$param := [I_{SC\_STC} = 9.79, U_{OC\_STC} = 24.27, m = 1, N\_C = 36, E_{STC} = 1000, E_G = 200, k = 1.380649000\ 10^{-23}, e = 1.602176634\ 10^{-19}, \vartheta_C = 25, \vartheta_{STC} = 25, TC_{I\_SC} = 0.05, TC_{U\_OC} = -0.29] \tag{2.1}$$

Main calculation

First the necessary quantities for the model of the PV generator and the starting values for the Jacobian matrix are calulated. Based on these the Jacobian matrix is determined and transformed, so that it can be used with the Newton-Raphson Method. Finally, the iterations are presented.

Necessary quantities

Thermal voltage:

```
> U__T := k * (vartheta__C + 273.15) / e;
```

$$U_T := \frac{k\,(\vartheta_C + 273.15)}{e} \tag{3.1.1}$$

Photocurrent:

```
> I__Ph := I__SC_STC * E__G/E__STC * (1 + TC__I_SC/100 *
  (vartheta__C - vartheta__STC));
```

$$I_{Ph} := \frac{I_{SC\_STC}\,E_G\left(1 + \frac{TC_{I\_SC}\,(\vartheta_C - \vartheta_{STC})}{100}\right)}{E_{STC}} \tag{3.1.2}$$

Photocurrent with constant solar irradiance $(E_G = E_{STC})$:

```
> I__Ph_const_irr := eval(I__Ph, E__G = E__STC);
```

$$I_{Ph\_const\_irr} := I_{SC\_STC}\left(1 + \frac{TC_{I\_SC}\left(\vartheta_C - \vartheta_{STC}\right)}{100}\right) \quad \textbf{(3.1.3)}$$

Open-circuit voltage with constant solar irradiance $\left(E_{\mathrm{G}} = E_{\mathrm{STC}}\right)$:

```
> U__OC_STC_const_irr := U__OC_STC * (1 + TC__U_OC/100 *
  (vartheta__C - vartheta__STC));
```

$$U_{OC\_STC\_const\_irr} := U_{OC\_STC}\left(1 + \frac{TC_{U\_OC}\left(\vartheta_C - \vartheta_{STC}\right)}{100}\right) \quad \textbf{(3.1.4)}$$

Starting values for the jacobian matrix

Starting value for the open circuit voltage:

```
> U__OC_0 := evalf(eval(U__OC_STC_const_irr + m * N_C * U__T *
  ln(E__G/E__STC), param)):
```

Starting value for the reverse saturation current:

```
> I__S_0 := evalf(eval(I__Ph * exp(- U__OC_0 / (m * N_C * U__T)
  ), param)):
```

Jacobian matrix

Preparing the vector of functions and zero crossings for Jacobian matrix:

```
> f__1 := exp( (U__OC_theta_phi - U__OC_STC_const_irr)/(m * N_C
  * U__T) ) - (I__Ph - I__S_theta)/(I__Ph_const_irr -
  I__S_theta):
> f__2 := I__S_theta - I__Ph * (exp( U__OC_theta_phi/(m * N_C *
  U__T) ) - 1)^(-1):
> f := < f__1, f__2 >:
> x__R := < I__S_theta, U__OC_theta_phi >:
```

Jacobian matrix:

```
> J := Jacobian(convert(f, list), convert(x__R, list)):
```

Preparing the Jacobian matrix for the Newton-Raphson method:

```
> J__inv := MatrixInverse(J):
> J__inv_num := eval(J__inv, param):
> f__num := eval(f, param):
```

Iterations

```
> x__R_0 := < I__S_0, U__OC_0 >;
```

$$x_{R\_0} := \begin{bmatrix} 3.935566032\ 10^{-11} \\ 22.78137801 \end{bmatrix} \quad \textbf{(3.4.1)}$$

```
> x__R_1 := evalf( x__R_0 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_0(1), U__OC_theta_phi = x__R_0(2)] ) );
```

$$x_{R\_1} := \begin{bmatrix} 3.93556604183892\ 10^{-11} \\ 22.7813780076877 \end{bmatrix} \quad \textbf{(3.4.2)}$$

```
> x__R_2 := evalf( x__R_1- eval( J__inv_num . f__num ,
  [I__S_theta = x__R_1(1), U__OC_theta_phi = x__R_1(2)] ) );
```

$$x_{R\_2} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \quad \textbf{(3.4.3)}$$

```
> x__R_3 := evalf( x__R_2- eval( J__inv_num . f__num ,
  [I__S_theta = x__R_2(1), U__OC_theta_phi = x__R_2(2)] ) );
```

$$x_{R\_3} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.4}$$

```
> x__R_4 := evalf( x__R_3 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_3(1), U__OC_theta_phi = x__R_3(2)] ) );
```

$$x_{R\_4} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.5}$$

```
> x__R_5 := evalf( x__R_4 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_4(1), U__OC_theta_phi = x__R_4(2)] ) );
```

$$x_{R\_5} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.6}$$

```
> x__R_6 := evalf( x__R_5 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_5(1), U__OC_theta_phi = x__R_5(2)] ) );
```

$$x_{R\_6} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.7}$$

```
> x__R_7 := evalf( x__R_6 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_6(1), U__OC_theta_phi = x__R_6(2)] ) );
```

$$x_{R\_7} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.8}$$

```
> x__R_8 := evalf( x__R_7 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_7(1), U__OC_theta_phi = x__R_7(2)] ) );
```

$$x_{R\_8} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.9}$$

```
> x__R_9 := evalf( x__R_8 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_8(1), U__OC_theta_phi = x__R_8(2)] ) );
```

$$x_{R\_9} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.10}$$

```
> x__R_10 := evalf( x__R_9 - eval( J__inv_num . f__num ,
  [I__S_theta = x__R_9(1), U__OC_theta_phi = x__R_9(2)] ) );
```

$$x_{R\_10} := \begin{bmatrix} 3.93556604183619\ 10^{-11} \\ 22.7813780046823 \end{bmatrix} \tag{3.4.11}$$