```python
# filename: heat_rod.py
import sympy as sp
import numpy as np

class SteadyStateRod:
    def __init__(self, a, b, h_val, k_val, Phi_val, Tamb_val, L):
        """
        Initialize the rod parameters.
        a, b: cross-section dimensions (cm)
        h_val: convective heat transfer coefficient (W/cm^2 C)
        k_val: thermal conductivity (W/cm C)
        Phi_val: source heat flux (W/cm^2)
        Tamb_val: ambient temperature (C)
        """
        # Store numeric values
        self.a_val = a
        self.b_val = b
        self.h_val = h_val
        self.k_val = k_val
        self.Phi_val = Phi_val
        self.Tamb_val = Tamb_val
        self.L = L

        # Define symbolic variables
        x, h, k, Phi, Tamb = sp.symbols('x h k Phi Tamb')
        self.x = x
        self.h = h
        self.k = k
        self.Phi = Phi
        self.Tamb = Tamb

        # Precompute gamma
        self.gamma = sp.sqrt(2*(a+b)*h/(a*b*k))

        # Define c1, c2
        c1 = -Phi/(k*self.gamma) * ((sp.exp(self.gamma*self.L)*(h + k*self.gamma)) / (sp.exp(-self.gamma*self.L)*(h - k*self.gamma) \
                                                                                       + sp.exp(self.gamma*self.L)*(h + k*self.gamma)))
        c2 = Phi/(k*self.gamma) + c1
        self.c1 = c1
        self.c2 = c2

        # Ts(x) symbolic
        Ts_expr = c1*sp.exp(-self.gamma*x) + c2*sp.exp(self.gamma*x) + Tamb
        self.Ts_expr = Ts_expr

        # Precompute derivatives symbolically
        self.dTs_dh_expr = sp.diff(Ts_expr, h)
        self.dTs_dk_expr = sp.diff(Ts_expr, k)
        self.dTs_dPhi_expr = sp.diff(Ts_expr, Phi)


        # Lambdify for fast numeric evaluation
        vars = (x, h, k, Phi, Tamb)
        self.Ts_func = sp.lambdify(vars, Ts_expr, 'numpy')
        self.dTs_dh_func = sp.lambdify(vars, self.dTs_dh_expr, 'numpy')
        self.dTs_dk_func = sp.lambdify(vars, self.dTs_dk_expr, 'numpy')
        self.dTs_dPhi_func = sp.lambdify(vars, self.dTs_dPhi_expr, 'numpy')

    def Ts(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        """Evaluate Ts at given x values. Use class defaults if not provided."""
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.Ts_func(x_vals, h, k, Phi, Tamb)

    def dTs_dh(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.dTs_dh_func(x_vals, h, k, Phi, Tamb)


    def dTs_dk(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.dTs_dk_func(x_vals, h, k, Phi, Tamb)

    def dTs_dPhi(self, x_vals, h=None, k=None, Phi=None, Tamb=None):
        h = h if h is not None else self.h_val
        k = k if k is not None else self.k_val
        Phi = Phi if Phi is not None else self.Phi_val
        Tamb = Tamb if Tamb is not None else self.Tamb_val
        return self.dTs_dPhi_func(x_vals, h, k, Phi, Tamb)
```