1 General Description

This repository provides code to compute the roots of single-variable functions, and nth-degree polynomials. The two primary subroutines are

1. `fun_zero()` - Used for finding roots of single-variable functions between a bounded interval.
2. `poly_roots()` - Used for finding the roots of nth-degree polynomials with constant coefficients.

2 Computing roots of single variable functions - `fun_zero()`

The function `fun_zero()` uses a combination of Brent's method, the secant method, and interpolation to quickly converge to the root of a single-variable function $f(x)$, provided the following conditions hold:

1. The function $f(x)$ is defined over the interval $[a, b]$.
2. There is a sign change over the interval $[a, b]$ (i.e., $f(a) \cdot f(b) < 0$).

2.1 Inputs

The function requires the following inputs:

1. `f_pointer` - A pointer to the function $f(x)$ to be evaluated.
2. `bounds` - A 2-element array containing the lower and upper bounds of the interval $[a, b]$.

2.2 Optional Inputs

1. `f0_options` - A derived type containing additional options:
   - `tolerance` - The stopping tolerance for the root-finding process.
   - `maxiter` - The maximum number of iterations allowed.
   - `show_iterations` - A logical flag indicating whether to print iteration details.
   - `show_results` - A logical flag indicating whether to print the final result.

2.3 Outputs

1. `status` - An integer indicating the status of the root-finding process:
   - `0` - The root was successfully found.
   - `1` - The interval does not bracket a root.
   - `2` - The maximum number of iterations was reached.
2. `root` - The computed (approximate) root of the function.

2.4 Optional Outputs

1. `niter` - The number of iterations required to find the root.

2.5 Error Handling

If the function $f(x)$ is undefined (NaN) at either bound a or b, an error message will be displayed, and the subroutine will terminate.

2.6 Example Usage with Options

An example usage of `fun_zero()`:

```fortran
program test
    use MOD_Select_Kind, only: pv
    use MOD_Roots
    implicit none

    real(pv) :: bounds(2), root
    integer :: status
    procedure(sv_function), pointer :: f_pointer => null()
    type(fun_zero_options) :: options

    bounds = [-2.0, 2.0]
    f_pointer => my_function  ! Pointer to the function to be solved

    options%show_iterations = .true.
    options%show_results    = .true.
    options%maxiter         = 100
    options%tolerance       = 1.0E-6_pv

    call fun_zero(f_pointer, bounds, status, root, f0_options = options)


    contains

    function my_function(x) result(y)
        real(pv), intent(in) :: x
        real(pv) :: y
        real(pv) :: pi

        pi = 3.141592653_pv
        y = -exp(-x) + 2.0 * x**2 / (exp(4.0*x)) - pi
    end function my_function


end program test
```

The output is

```
    Iteration:    1     f(x) = -3.2742E+00
    Iteration:    2     f(x) = -3.2743E+00
    Iteration:    3     f(x) = -4.1419E+00
    Iteration:    4     f(x) = -4.1419E+00
    Iteration:    5     f(x) = -1.0900E+00
    Iteration:    6     f(x) = -1.0900E+00
    Iteration:    7     f(x) = -1.0900E+00
    Iteration:    8     f(x) = -1.0900E+00
    Iteration:    9     f(x) = -1.0948E-01
    Iteration:   10     f(x) = -1.0948E-01
    Iteration:   11     f(x) = -1.0948E-01
    Iteration:   12     f(x) =  2.9326E-02
    Iteration:   13     f(x) =  2.9326E-02
    Iteration:   14     f(x) = -5.7377E-03
    Iteration:   15     f(x) = -1.6654E-05
    Iteration:   16     f(x) = -1.6654E-05
    Iteration:   17     f(x) = -1.6654E-05
    Iteration:   18     f(x) = -1.6654E-05
    Iteration:   19     f(x) = -1.6654E-05
    Iteration:   20     f(x) = -1.6654E-05
```

```
    Iteration:   21     f(x) = -1.6654E-05
    Iteration:   22     f(x) = -1.6654E-05
    Iteration:   23     f(x) = -1.6654E-05
    Iteration:   24     f(x) = -1.6654E-05
    Iteration:   25     f(x) =  1.1917E-05

    Root found: -5.34542E-01 after          25
     iterations.
     Status:            0
```