User Guide: `F90-Extrapolation-Integration`

Álvaro R. Puente-Uriona

February 16, 2023

1 Introduction

`F90-Extrapolation-Integration` is a module written in `FORTRAN90`, which specializes in computing integrals of the type

$$I(\alpha) = \int_{\boldsymbol{a}}^{\boldsymbol{b}} d^d\boldsymbol{x}\; f\left(\boldsymbol{x};\alpha\right), \tag{1}$$

for $d = 1, 2, 3$, where the d-dimensional variable $\boldsymbol{x}$ is defined over $\mathbb{R}^d$ or $\mathbb{C}^d$ and within the d-dimensional bounds $\boldsymbol{a}$, $\boldsymbol{b}$. The general index α denotes all other possible dependences of f other than $\boldsymbol{x}$. Integrals as Eq. (1) are a common occurrence in scientific computing and the `F90-Extrapolation-Integration` module specializes in providing an adaptive method, based on extrapolation over the *trapezium method*, described in Ref. [1], to obtain increasingly accurate values for $I(\alpha)$, dependent only in the number of sample points in which f is evaluated.

The project is hosted at `https://github.com/irukoa/F90-Extrapolation-Integration`.

2 Definitions

- We call Eq. (1) a *vector* integral.
- A *scalar* integral is an special case of Eq. (1), in which I and f do not depend on any parameter α.
- An array `m(:)` in *memory layout* is a 1-dimensional array, defined from a d-dimensional array with arbitrary bounds `a(:, :, ..., :)`, which we call array in *arbitary layout*. `m` has the form,

  ```
  real(kind=dp)    :: m(1:size(a))
  complex(kind=dp) :: m(1:size(a))
  ```

  where the lower bound is always 1 and the upper bound `size(a)`. The mapping between `a` and `m` is done *via*

  ```
  count = 1
  do i1 = lbound(a, 1), ubound(a, 1)
    do i2 = lbound(a, 2), ubound(a, 2)
     .
      .
       .
        do id = lbound(a, d), ubound(a, d)
  ```

```
        m(count) = a(i1, i2, ..., id) !From arbitrary to memory layout.
        a(i1, i2, ..., id) = m(count) !From memory to arbitrary layout.
        count = count + 1
      enddo
     .
    .
   .
  enddo
enddo
```

- A *discretization of dimension* j corresponds to the partition of the variable x_j in Eq. (1) within the $[a_j, b_j]$ range, given by

$$x_j\left\{i_j\right\} = a_j + (i_j - 1)\, w_j, \quad i_j \in [1, N_j],\ w_j = \frac{b_j - a_j}{N_j - 1}. \tag{2}$$

- A *discretization point in memory layout* is an index `count` representing a d-dimensional point $(x_1\{i_1\}, \cdots, x_d\{i_d\})$ *via*

```
count = 1
do i1 = 1, N1
  do i2 = 1, N2
   .
    .
     .
      do id = 1, Nd
        !{i1, i2, ..., id} are identified with count.
        count = count + 1
      enddo
     .
    .
   .
  enddo
enddo
```

3 Module Overview

3.1 Integration-Extrapolation Routines

`integral_extrapolation` is a family of routines, which is called

```
call integral_extrapolation(array, sizes, int_bounds, result, info)
```

with

```
real/complex(kind=dp), intent(in)  :: array(:), int_bounds(:)
integer,               intent(in)  :: sizes(:)

real/complex(kind=dp), intent(out) :: result
integer,               intent(out) :: info
```

for scalar integrals. The input/output variables are

- `array(:)` is a complex or real `size(array) = prod(sizes)` array in memory layout, where each index represents the discretization point $(x_1\{i_1\}, \cdots, x_d\{i_d\})$ in memory layout and the array contains the data of the integrand $f(x_1\{i_1\}, \cdots, x_d\{i_d\})$ evaluated in each discretization point.

- `sizes(:)` is an integer `size = 1`, `size = 2` or `size = 3` array containing the number of discretization points in each dimension (N_j in Eq. (2) for $j = 1, 2, 3$). The routine employs the method to compute a 1-dimensional, 2-dimensional or 3-dimensional integral depending on `size(sizes)`. To use extrapolation, all of the integers in the array must be expressible as $2^n + 1$ for some $n \in 0, 1, \cdots$. The only exception is `sizes(j) = 1` for some $j = 1, 2, 3$. In that case, the integral in dimension j is set to be $b_j - a_j$. In all the cases where `sizes(j)` is not 1 and can not be expressed as $2^n + 1$, the *rectangle method* [1] is used for the integration.

- `int_bounds(:)` is a `size = 2*size(sizes)` real or complex array (not necesarily `kind = array`) which contains the integration bounds of Eq. (1) sorted in ascending dimension containing the lower bound and the upper bound respectively. For example, for a $d = 2$ integral with $\boldsymbol{a} = (0, 1)$, $\boldsymbol{b} = (2, 4)$,

  `int_bounds = (/0.0_dp, 2.0_dp, 1.0_dp, 4.0_dp/).`

- `result` is a `kind = array` complex or real number containing the scalar integral I in the cases `info = 0, 1` and is initialized to 0 in the case `info = -1`.

- `info` is an integer, reporting the calculation status:
  - `info = 1`: Calculation successful and `result` contains the integral computed using extrapolation methods.
  - `info = 0`: Calculation successful and `result` contains the integral computed using the rectangle method.
  - `info = -1`: Error. Returning `result = 0`.

For vector integrals the input/output variables are slightly different,

```
real/complex(kind=dp), intent(in)  :: array(:, :), int_bounds(:)
integer,               intent(in)  :: sizes(:)

real/complex(kind=dp), intent(out) :: result(:)
integer,               intent(out) :: info
```

Where `sizes(:)`, `int_bounds(:)` and `info` are the same as for a scalar integral. However,

- `array(:, :)` contains:
  - In the first dimension, the same information as for scalar integrals.
  - In the second dimension, an index representing α in Eq. (1), in memory layout, which will not be integrated over.

- `result(:)` inherits the second dimension of `array(:, :)`, thus containing an index representing α in memory layout.

3.2 Shrink Array Routines

`shrink_array` is a family of routines, used to pass arrays from arbitrary layout to memory layout, which is called

```
call shrink_array(array, shrink, info)
```

with the following possibilities for `array`,

```
real/complex(kind=dp), intent(in)  :: array(:)
real/complex(kind=dp), intent(in)  :: array(:, :)
real/complex(kind=dp), intent(in)  :: array(:, :, :)
real/complex(kind=dp), intent(in)  :: array(:, :, :, :)
```

and

```
real/complex(kind=dp), intent(out) :: shrink(:)
integer,               intent(out) :: info
```

The input/output variables are

- `array(:)`, `array(:, :)`, `array(:, :, :)`, `array(:, :, :, :)` is a complex or real array with arbitrary bounds in each dimension.
- `shrink(:)` is a `kind = array` complex or real array, which contains `array` in memory layout.
- `info` is an integer, reporting the calculation status:
  - `info = 1`: Calculation successful and `shrink` contains `array` in memory layout.
  - `info = -1`: Error. Returning `shrink = 0`.

3.3 Expand Array Routines

`expand_array` is a family of routines, used to pass arrays from memory layout to arbitrary layout, which is called

```
call expand_array(array, expand, info)
```

with

```
real/complex(kind=dp), intent(in)  :: array(:)
integer,               intent(out) :: info
```

and the following possibilities for `expand`,

```
real/complex(kind=dp), intent(out) :: expand(:)
real/complex(kind=dp), intent(out) :: expand(:, :)
real/complex(kind=dp), intent(out) :: expand(:, :, :)
real/complex(kind=dp), intent(out) :: expand(:, :, :, :)
```

The input/output variables are

- `array(:)` is a complex or real array in memory layout.
- `expand(:)`, `expand(:, :)`, `expand(:, :, :)`, `expand(:, :, :, :)` is `kind = array` a complex or real array with arbitrary bounds in each dimension into which `array(:)` is to be casted.
- `info` is an integer, reporting the calculation status:
  - `info = 1`: Calculation successful and `expand` contains `array` in arbitrary layout.
  - `info = -1`: Error. Returning `expand = 0`.

4 Example

We provide an example in the file `example.F90`. The objective is to calculate

$$
\begin{aligned}
I(v) = \int_0^2 dx \int_0^2 dy \int_0^2 dz \left[\cos(x)e^{\sin(vx)} + i\cos(x)e^{\sin(2vx)}\right] \times \\
\left[\cos(y)e^{\sin(vy)} + i\cos(y)e^{\sin(2vy)}\right]\left[\cos(z)e^{\sin(vz)} + i\cos(z)e^{\sin(2vz)}\right],
\end{aligned}
\tag{3}
$$

for $v = -1, 0, 1$. To do this, we consider a set of integers `l1, l2, l3` into which we discretize each dimension and obtain the values of the integrand in each discretization point for the considered v-s. After gathering the data, we pass the index related to v to memory layout. The integration is the performed by `integral_extrapolation`. Finally, the program prints $I(-1)$, which is known to be exactly $-0.0481480+0.352825i$. By default, `l1 = l2 = l3` $= 2^5+1 = 33$, so the extrapolation method is employed. The user is encouraged to give different values for `l1, l2, l3`, specially some not expressible as $2^n + 1$, such as `l1 = l2 = l3` $= 100$. This way, the provided result will be estimated by the rectangle approximation rather than by the extrapolation method.

References

[1] Álvaro R. Puente-Uriona. In preparation, 2023.