ecx

Release 0.1.0dev0

Milan Skocic

Jan 07, 2026

CONTENTS

Electrochemistry for Fortran.

CHAPTER

ONE

GETTING STARTED

1.1 Introduction

ecx a Fortran library for providing a collection of routines for electrochemistry. A C API allows usage from C, or can be used as a basis for other wrappers. A Python wrapper allows easy usage from Python.

To use *ecx* within your fpm project, add the following lines to your file:

```
[dependencies]
ecx = { git="https://github.com/MilanSkocic/ecx.git" }
```

1.2 Dependencies

```
gcc>=10
gfortran>=10
fpm>=0.7
stdlib>=0.7
```

1.3 Installation

A Makefile is provided, which uses fpm, for building the library.

- On windows, msys2 needs to be installed. Add the msys2 binary (usually C:\msys64\usr\bin) to the path in order to be able to use make.
- On Darwin, the gcc toolchain needs to be installed.

```
chmod +x configure.sh
./configure.sh
make
make test
make install
make uninstall
```

You need a compiler that can compile the stdlib.

1.4 License

MIT

CHAPTER

TWO

EXAMPLES

2.1 Fortran

```
program example_in_f
    use iso_fortran_env
    use ecx
    implicit none

    real(real64) :: w(3) = [1.0d0, 1.0d0, 100.0d0]
    real(real64) :: r = 100.0d0
    real(real64) :: p(3) = 0.0d0
    character(len=1) :: e
    integer :: errstat
    complex(real64) :: zout(3)
    character(len=:), pointer :: errmsg

    p(1) = r
    e = "R"
    call z(p, w, zout, e, errstat, errmsg)
    print *, zout
    print *, errstat, errmsg


end program
```

2.2 C

```
#include <stdio.h>
#include <stdlib.h>
#include "ecx.h"


int main(void){

    int errstat, i;
    double w[3] = {1.0, 1.0, 1.0};
    double p[3] = {100.00, 0.0, 0.0};
    ecx_cdouble z[3] = {ecx_cbuild(0.0,0.0),
                        ecx_cbuild(0.0, 0.0),
```

(continues on next page)

(continued from previous page)

```
                        ecx_cbuild(0.0, 0.0)};
    char *errmsg;

    ecx_eis_z(p, w, z, 'R', 3, 3, &errstat, &errmsg);

    for(i=0; i<3;i++){
        printf("%f %f \n", creal(z[i]), cimag(z[i]));
    }
    printf("%d %s\n", errstat, errmsg);
    return EXIT_SUCCESS;
}
```

2.3 Python

```
from pyecx import eis
import matplotlib.pyplot as plt

R = 100
C = 1e-6
w = np.logspace(6, -3, 100)

p = np.asarray([R, 0.0, 0.0])
zr = np.asarray(eis.z("R", w, p))
p = np.asarray([C, 0.0, 0.0])
zc = np.asarray(eis.z("C", w, p))
zrc = zr*zc / (zr+zc)
print("finish")

fig = plt.figure()
ax = fig.add_subplot(111)

ax.set_aspect("equal")
ax.plot(zrc.real, zrc.imag, "g.", label="R/C")

ax.invert_yaxis()

plt.show()
```

CHAPTER

THREE

APIS

3.1 Fortran

https://milanskocic.github.io/ecx/ford/index.html

3.1.1 ecx (module)

Main module for ecx.

3.1.2 ecx__api (module)

API Procedures

get_version (function)

```
function get_version() -> fptr
```

Get the version.

Returns

fptr
[character(len=:), pointer] Version of the library.

kTe (function)

```
pure elemental function kTe(T) -> r
```

Compute the thermal voltage.

Arguments

T
[real(dp), intent(in)] Temperature in °C.

Returns

r
[real(dp)] Thermal voltage in V.

z (subroutine)

```
subroutine z(p, w, zout, e, errstat, errmsg)
```

Arguments

p
[real(dp), intent(in), dimension(:)]

w
[real(dp), intent(in), dimension(:)] Angular frequencies in rad.s-1

zout
[complex(dp), intent(out), dimension(:)] Complex impedance in Ohms.

e
[character(len=1), intent(in)] Electrochemical element: R, C, L, Q, O, T, G

errstat
[integer(int32), intent(out)] Error status

errmsg
[character(len=:), intent(out), pointer] Error message

mm (subroutine)

```
subroutine mm(p, w, zout, n)
```

Arguments

p
[real(dp), intent(in), dimension(:)] Compute the measurement model.

w
[real(dp), intent(in), dimension(:)] Parameters.

zout
[complex(dp), intent(out), dimension(:)] Angular frequencies in rad.s-1

n
[integer(int32), intent(in)] Complex impedance in Ohms.

nernst (function)

```
pure function nernst(E0, z, aox, vox, ared, vred, T) -> E
```

Compute the Nernst electrochemical potential in V.

Arguments

E0
[real(dp), intent(in)]

z
[integer(int32), intent(in)] Standard electrochemical potential in V.

aox
[real(dp), intent(in), dimension(:)] Number of exchanged electrons.

vox
[real(dp), intent(in), dimension(:)] Activities of the oxidants.

ared
[real(dp), intent(in), dimension(:)] Coefficients for the oxidants.

vred
[real(dp), intent(in), dimension(:)] Activities of the reductants

T
[real(dp), intent(in)] Coefficients for the reductants.

Returns

E
[real(dp)] Temperature in °C.

sbv (function)

```
pure elemental function sbv(U, OCV, j0, aa, ac, za, zc, A, T) -> I
```

Arguments

U
[real(dp), intent(in)] Open Circuit Voltage in V.

OCV
[real(dp), intent(in)] Compute Butler Volmer equation without mass transport.

j0
[real(dp), intent(in)] Electrochemical potential in V.

aa
[real(dp), intent(in)] Exchange current density in A.cm-2.

ac
[real(dp), intent(in)] Anodic transfer coefficient.

za
[real(dp), intent(in)] Cathodic transfer coefficient.

zc
[real(dp), intent(in)] Number of exchnaged electrons in the anodic branch.

A
[real(dp), intent(in)] Number of exchnaged electrons in the cathodic branch.

T
[real(dp), intent(in)] Area in cm2.

Returns

I
[real(dp)] Temperature in °C.

bv (function)

```
pure elemental function bv(U, OCV, j0, jdla, jdlc, aa, ac, za, zc, A, T) -> I
```

Compute Butler Volmer equation with mass transport.

Arguments

U
[real(dp), intent(in)] Open Circuit Voltage in V.

OCV
[real(dp), intent(in)]

j0
[real(dp), intent(in)] Electrochemical potential in V.

jdla
[real(dp), intent(in)] Exchange current density in A.cm-2.

jdlc
[real(dp), intent(in)] Anodic diffusion limiting current density in A.cm-2.

aa
[real(dp), intent(in)] Cathodic diffusion limiting current density in A.cm-2.

ac
[real(dp), intent(in)] Anodic transfer coefficient.

za
[real(dp), intent(in)] Cathodic transfer coefficient.

zc
[real(dp), intent(in)] Number of exchnaged electrons in the anodic branch.

A
[real(dp), intent(in)] Number of exchnaged electrons in the cathodic branch.

T
[real(dp), intent(in)] Area in cm2.

Returns

I
[real(dp)] Temperature in °C.

3.1.3 ecx__capi (module)

Procedures

capi_get_version (function)

```
function capi_get_version()bind(c, name="ecx_get_version") -> cptr
```

C API - Get the version.

Returns

cptr
[type(c_ptr)]

capi_nm2eV (subroutine)

```
pure subroutine capi_nm2eV(lambda, E, n)bind(C, name="ecx_core_nm2eV")
```

Arguments

lambda
[real(c_double), intent(in), dimension(n)] Wavelength in nm.

E
[real(c_double), intent(out), dimension(n)] Energy in eV.

n
[integer(c_size_t), intent(in), value] Size of lambda and E.

capi_kTe (subroutine)

```
pure subroutine capi_kTe(T, kTe_, n)bind(C, name="ecx_core_kTe")
```

Arguments

T
[real(c_double), intent(in), dimension(n)] Temperature in °C.

kTe_
[real(c_double), intent(out), dimension(n)] Thermal voltage in V.

n
[integer(c_size_t), intent(in), value] Size of T and kTe.

capi_z (subroutine)

```
subroutine capi_z(p, w, zout, e, k, n, errstat, errmsg)bind(C, name="ecx_eis_z")
```

Arguments

p
[real(c_double), intent(in), dimension(k)] Parameters.

w
[real(c_double), intent(in), dimension(n)] Angular frequencies in rad.s-1

zout
[complex(c_double_complex), intent(out), dimension(n)] Complex impedance in Ohms.

e
[character(len=1,kind=c_char), intent(in), value] Electrochemical element: R, C, L, Q, O, T, G

k
[integer(c_size_t), intent(in), value] Size of p

n
[integer(c_size_t), intent(in), value] Size of w

errstat
[integer(c_int), intent(out)] Error status

errmsg
[type(c_ptr), intent(out)] errmsg Error message

capi_nernst (function)

```
pure function capi_nernst(E0, z, aox, vox, nox, ared, vred, nred, T)bind(C,
name="ecx_kinetics_nernst") -> E
```

Compute the Nernst electrochemical potential in V.

Arguments

E0
[real(c_double), intent(in), value]

z
[integer(c_int), intent(in), value] Standard electrochemical potential in V.

aox
[real(c_double), intent(in), dimension(nox)] Number of reductants.

vox
[real(c_double), intent(in), dimension(nox)] Activities of the oxidants.

nox
[integer(c_size_t), intent(in), value] Number of exchanged electrons.

ared
[real(c_double), intent(in), dimension(nred)] Coefficients for the oxidants.

vred
[real(c_double), intent(in), dimension(nred)] Activities of the reductants

nred
[integer(c_size_t), intent(in), value] Number of oxidants.

T
[real(c_double), intent(in), value] Coefficients for the reductants.

Returns

E
[real(c_double)] Temperature in °C.

capi_sbv (subroutine)

```
pure subroutine capi_sbv(U, OCV, j0, aa, ac, za, zc, A, T, I, n)bind(c,
name="ecx_kinetics_sbv")
```

Compute Butler Volmer equation without mass transport.

Arguments

U
[real(c_double), intent(in), dimension(n)] Open circuit potential in volts.

OCV
[real(c_double), intent(in), value] Size of U and I.

j0
[real(c_double), intent(in), value] Potential in volts.

aa
[real(c_double), intent(in), value] Exchange current density in A.cm-2.

ac
[real(c_double), intent(in), value] Anodic transfert coefficient.

za
[real(c_double), intent(in), value] Cathodic transfert coefficient.

zc
[real(c_double), intent(in), value] Number of exchanged electrons in anodic branch.

A
[real(c_double), intent(in), value] Number of exchanged electrons in cathodic branch.

T
[real(c_double), intent(in), value] Area in cm2.

I
[real(c_double), intent(out), dimension(n)] Temperature in °C.

n
[integer(c_size_t), intent(in), value]

capi_bv (subroutine)

```
pure subroutine capi_bv(U, OCV, j0, jdla, jdlc, aa, ac, za, zc, A, T, I, n)bind(c,
name="ecx_kinetics_bv")
```

Compute Butler Volmer equation without mass transport.

Arguments

U
[real(c_double), intent(in), dimension(n)] Open circuit potential in volts.

OCV
[real(c_double), intent(in), value] Size of U and I.

j0
[real(c_double), intent(in), value] Potential in volts.

jdla
[real(c_double), intent(in), value] Exchange current density in A.cm-2

jdlc
[real(c_double), intent(in), value] Anodic diffusion limiting current density in A.cm-2.

aa
[real(c_double), intent(in), value] Cathodic diffusion limiting current density in A.cm-2.

ac
[real(c_double), intent(in), value] Anodic transfert coefficient.

za
[real(c_double), intent(in), value] Cathodic transfert coefficient.

zc
[real(c_double), intent(in), value] Number of exchanged electrons in anodic branch.

A
[real(c_double), intent(in), value] Number of exchanged electrons in cathodic branch.

T
[real(c_double), intent(in), value] Area in cm2.

I
[real(c_double), intent(out), dimension(n)] Temperature in °C.

n
[integer(c_size_t), intent(in), value]

3.2 C

```c
#ifndef ECX_H
#define ECX_H
#include <complex.h>
#if _MSC_VER
    #define ADD_IMPORT __declspec(dllimport)
    typedef _Dcomplex ecx_cdouble;
    #define ecx_cbuild(real, imag) (_Cbuild(real, imag))
#else
    #define ADD_IMPORT
    typedef double _Complex ecx_cdouble;
    #define ecx_cbuild(real, imag) (real+I*imag)
#endif

extern char* ecx_get_version(void);


ADD_IMPORT extern const double ecx_core_PI;
```

(continues on next page)

(continued from previous page)

```
ADD_IMPORT extern const double ecx_core_T_K;
void ecx_core_nm2eV(double *lambda, double *E, size_t n);
void ecx_core_kTe(double *U, double *kTE, size_t n);


extern double ecx_kinetics_nernst(double E0, int z,
                                  double *aox, double *vox, size_t nox,
                                  double *ared, double *vred, size_t nred,
                                  double T);

extern void ecx_kinetics_sbv(double *U, double OCV, double j0,
                             double aa, double ac, double za, double zc,
                             double A, double T, double *i, size_t n);

extern void ecx_kinetics_bv(double *U, double OCV, double j0, double jdla, double jdlc,
                        double aa, double ac, double za, double zc,
                        double A, double T, double *i, size_t n);


extern void ecx_eis_z(double *p, double *w, ecx_cdouble *z,
                      char e, size_t k, size_t n,
                      int *errstat, char *(*errmsg));

#endif
```

3.3 Python

Python wrapper of the (Modern Fortran) ecx library.

CHAPTER

FOUR

PROGRAMS

4.1 ecxcli (program)

```
program ecxcli
    !! CLI interface for electrochemistry.
    use ieee_arithmetic, only: ieee_is_nan
    use iso_fortran_env, only: real64, int32, output_unit
    use ecx
    use ciaaw, only: ciaaw_version => get_version, get_saw, print_periodic_table,&
                     get_z_by_symbol
    use M_CLI2
    implicit none

    integer :: i
    integer(int32) :: zmass
    real(real64) :: r, dr
    character(len=32) :: cmd
    character(len=:),allocatable, target  :: help_text(:), version_text(:), usage_text(:)
    character(len=:), pointer :: char_fp(:)
    character(len=3) :: elmt

    nullify(char_fp)

    help_text=[character(len=80) :: &
        'NAME                                                            ', &
        '  ecxcli(1) - Command line for ecx                              ', &
        '                                                                ', &
        'SYNOPSIS                                                        ', &
        '  ecxcli SUBCOMMAND [OPTIONS ...] ARGS ...                      ', &
        '                                                                ', &
        'DESCRIPTION                                                     ', &
        '  ecxcli is command line interface for computing electro-       ', &
        '  chemical properties:                                          ', &
        '    o EIS         Electrochemical Impedance Z=f(w)             ', &
        '    o Kinetics    j=f(U)                                       ', &
        '    o PEC         Iph=f(hv, U)                                 ', &
        '                                                               ', &
        '  It can also provide the molar masses, isotope compositions and', &
        '  nuclide compositions.', &
        '                                                                ', &
        'SUBCOMMANDS                                                     ', &
```

(continues on next page)

(continued from previous page)

```
        '  o all  Get the whole periodic table.                          ', &
        '  o saw  Get the standard atomic weight.                        ', &
        !' o ice  Get the isotopic composition of the element.           ', &
        !' o naw  Get the nuclide atomic weight.                         ', &
        '                                                                ', &
        '  Enter ecxcli SUBCOMMAND --help for detailed descriptions.     ', &
        '                                                                ', &
        'OPTIONS                                                         ', &
        '  o --abridged, -a     Use the abridged value.                  ', &
        '  o --uncertainty, -u  Use the uncertainty.                     ', &
        '  o --pprint           Nice formatting.                         ', &
        '  o --mass, -z         Get the mass number.                     ', &
        '                                                                ', &
        'VALID FOR ALL SUBCOMMANDS                                       ', &
        '  o --help     Show help text and exit                          ', &
        '  o --verbose  Display additional information when available.   ', &
        '  o --version  Show version information and exit.               ', &
        '                                                                ', &
        '' ]

    version_text=[character(len=80) :: &
        'PROGRAM:      ecxcli                ', &
        'DESCRIPTION:  Command line for ecx  ', &
        'VERSION:      '//get_version()//'   ', &
        'AUTHOR:       M. Skocic             ', &
        'LICENSE:      MIT                   ', &
        '' ]

    usage_text=[character(len=80) :: &
        'Usage: ecxcli SUBCOMMAND [OPTIONS...]|[--help|--version] ELEMENTS...', &
        '' ]

    cmd = get_subcommand()
    call set_mode('strict')

    select case (cmd)
        case ("saw")
            call set_args("--abridged:a --uncertainty:u --mass:z --pprint", get_help_
↪saw(), version_text)
            if(size(unnamed) == 1) then
                write(output_unit, "(A)") "Enter at least one element."
                char_fp => usage_text
                call print_text(char_fp)
                stop
            end if
            do i=2, size(unnamed)
                elmt = unnamed(i)
                zmass = get_z_by_symbol(elmt)
                if(lget("pprint"))then
                    r = get_saw(elmt,abridged=lget("a"))
                    dr = get_saw(elmt, abridged=lget("a"), uncertainty=.true.)
                    if(lget("mass"))then
```

(continues on next page)

(continued from previous page)

```
                        write(output_unit, '(A4, A3, A3, SP, G14.6, A3, ES14.2, S, A5,␣
↪I3, A1)') &
                              'SAW_', elmt, " = ", r, "+/-", dr, ' (Z=', zmass, ')'
                    else
                        write(output_unit, '(A4, A3, A3, SP, G14.6, A3, ES14.2)') &
                              'SAW_', elmt, " = ", r, "+/-", dr
                    end if
                else
                    r = get_saw(elmt,abridged=lget("a"), uncertainty=lget("u"))
                    if(lget("mass"))then
                        write(output_unit, '(I4, 4X, G0.16)') zmass, r
                    else
                        write(output_unit, '(G0.16)') r
                    end if
                end if
            end do
        case ("all")
            call set_args(" ", help_text, version_text)
            call  print_periodic_table()
        case default
            call set_args(" ", help_text, version_text)
            char_fp => usage_text
            call print_text(char_fp)
    end select

contains

function get_help_saw()result(res)
    !! Get the help text for the subcommand saw.
    character(len=80), allocatable :: res(:)
    res = [character(len=80) :: &
        'NAME                                                              ', &
        '  saw - th ecxcli subcommand to get the standard atomic weight. ', &
        '                                                                  ', &
        'SYNOPSIS                                                          ', &
        '  ecxcli saw [OPTIONS ...] ELEMENTS ...                        ', &
        '                                                                  ', &
        'DESCRIPTION                                                       ', &
        '  Provide the saw either abridged or not.                         ', &
        '  The uncertainty of the saw can be retrieved too.                ', &
        '                                                                  ', &
        'OPTIONS                                                           ', &
        '  o --abridged, -a     Use the abridged value.                    ', &
        '  o --uncertainty, -u  Use the uncertainty.                       ', &
        '  o --pprint           Nice formatting.                           ', &
        '  o --mass, -z         Get the mass number.                       ', &
        '                                                                  ', &
        'VALID FOR ALL SUBCOMMANDS                                         ', &
        '  o --help     Show help text and exit                           ', &
        '  o --verbose  Display additional information when available.    ', &
        '  o --version  Show version information and exit.                ', &
        '                                                                  ', &
```

(continues on next page)

(continued from previous page)

```
        'EXAMPLE                                                        ', &
        '  Minimal example                                              ', &
        '       ecxcli saw H                                            ', &
        '       ecxcli saw -a -u O                                      ', &
        '       ecxcli saw H C B O Zr Nb --pprint                       ', &
        '       ecxcli saw -a H C B O Zr Nb --pprint                    ', &
        '                                                               ', &
        '' ]
end function

subroutine print_text(char_fp)
    character(len=:), pointer, intent(in) :: char_fp(:)
    integer :: i
    do i=1, size(char_fp), 1
        write (OUTPUT_UNIT, '(A)') char_fp(i)
    end do
end subroutine


end program
```

Dependencies

- M_CLI2
- ciaaw
- *ecx*
- ieee_arithmetic
- iso_fortran_env

Procedures

4.1.1 get_help_saw (function)

`function get_help_saw() -> res`

Get the help text for the subcommand saw.

Returns

res
[character(len=80), allocatable, dimension(:)]

4.1.2 print_text (subroutine)

`subroutine print_text(char_fp)`

Arguments

char_fp
[character(len=:), pointer, intent(in), dimension(:)]

CHAPTER

FIVE

CHANGELOG

5.1 Version 0.1.0-dev

- Implementation of eis + C API
- Python wrappers for eis.

PYTHON MODULE INDEX

p

INDEX