***Electrochemistry: Advanced Modeling and Simulations, AGH, Kraków.***
Notes for the Project, part 2 (Comments on the source code and program)
Department of Solid State Chemistry

## Input and output data

The resulting system of ordinary differentia equations (ODEs) exhibits stiffness so special integrator is needed. In our case we will is the C++ implementation of RADAU5, which originally was developed by Hairer and Wanner in Fortran. The project for Dev-C++ environment was prepared and is included in the *npp, BB.zip* file. Download and unpack this file to any folder, and next click on the *npp, BB.dev* file. It will open in Dev-C++ environment (provided it is installed on your computer) and is ready to use. After any changes you make the project should be recompiled (for example by selecting *Execute | Rebuilt All* or by using shortcut *Ctrl+F11*). This produces an executable file *npp, BB.exe* in the same folder and you can run this program from the command line window. The output of the program consists of two files

*res-c(x,t).txt* – contains the concentration profiles for selected times

*res-mempot(t).txt* – contains the membrane potential for selected times

and the membrane potential is also displayed in the window (together with some additional information such as grid distances or initial values – just for checking purposes).

The profiles in space variable $x \in [0, d]$ are stored in *res-c(x,t).txt* file in rows. Each row is preceded by the time information and next there are $r$ rows (one for every component). Every row contain exactly $n+1$ values (for each node in the grid) separated by spaces. Here is an example for three component simulation ($r = 3$)

```
…
t=4.00:
0.004850 0.004941 0.005006 0.005053 0.005087 0.005853 0.006184
0.005164 0.005166 0.005167 0.005168 0.005168 0.005768 0.007158
0.005169 0.005169 0.005173 0.005170 0.005184 0.006370 0.006531
t=6.00:
…
```

The membrane potential as a function of time is stored in the *res-mempot(t).txt* file in two columns: the first contains selected times and the second corresponding potential. Values are separated by spaces.

Output to the file is carried out by the classical C language methods. First the file pointer is declared:

```
FILE *fp;
```

and next the connection with a disk file is established by the function `fopen()`. Specifically the instruction

```
fp = fopen("res-c(x,t).txt", "wt");
```

creates the file named *res-c(x,t).txt* in the text mode. This means the numbers stored in this file will be as string of digits (so directly human-readable). If for some reasons the creation fails the function `fopen()` returns the empty (null) pointer. That is way the opening the file is performed by the standard C language idiom

```
    if ((fp = fopen("res-c(x,t).txt", "wt")) == NULL) {
        cout << "Error on opening the file. I'am quiting.";
        return (1);
    }
```

Actual saving to the files is performed in the special part of the `IntegratorT` class. It is done by modification of the method `SolutionOutput` in this class. Implementation is in the file *IntegratorT.cpp*. However one must be careful with the names of variables. Namely the identifier $n$ in this class means the total number of ODE equations that are integrated – not a number of grid intervals (as we denote it in the main function). Thus we have to recalculate the number of grid intervals form the total number of equations form the equality:

$$\text{num\_of\_eqs} = (r+1)\cdot(\text{num\_of\_intervals} + 1) + 1$$

hence

$$\text{num\_of\_intervals} = (\text{num\_of\_eqs} - 1)/(r+1) - 1$$

In the method `SoultionOutput` the variable `lastNode` is used to denote number of intervals so the above expression taks the form

$$\text{lastNode} = (n-1)/(r+1) - 1$$

where – we stress once again – $n$ means the number of all equations (not grid number!).

Data for the simulations are embedded in the code so every time you change them the project must be recompiled. These data are contained in two files: *main.cpp* and *constants.h*.

The *constants.h* file:

n – the number of grid intervals, so n+1 is the number of grid points,

r – the number of species (components),

$R$, $T$, $\varepsilon_0$, $\varepsilon$ – have their usual meaning.

In the *main.cpp* there are:

D[r] – the table for diffusion coefficients

cL[r], cR[r] – the tables for left and right bulk concentrations,

cM[r], cM[r] – the tables for initial concentrations in the membrane according to the formula,

$$c_i(x,0) = \begin{cases} cML[i] & \text{for } 0 \le x < \tfrac{1}{2}d, \\ \tfrac{1}{2}(cML[i] + cMR[i]) & \text{for } x = \tfrac{1}{2}d, \\ cMR[i] & \text{for } \tfrac{1}{2}d < x \le d. \end{cases} \tag{0.1}$$

kLf[r], kLb[r], kRf[r], kRb[r] – the tables for heterogeneous rate constants on the left and right boundary (*f* for forward, *b* for backward),

z[r] – the table for components charges (valences). If we set $z[i] = 0,$ then the *i-th* species is not an ion.

xend – the duration of process (defines the period of time over which the process is carried out)

dx – time interval for the results output

All values are consistently expressed in SI units. Thus for example $cL[0]=100$ means $100\ mol/m^3$, hence it is $0.1M$.

## Grids generation

Generally we cannot rely on the uniform grid because in most simulated systems there are greater concentration variations close to the boundaries or in the middle of the membrane. Three functions – called *hGridGen1*, *hGridGen2*, *hGridGen3* – are provided. The implementation code is in a separate file *grids.cpp* included in the project. In each case the grid is produced in the standard unit interval $[0, 1]$. Next, outside of these functions, this standard grid is mapped on the rescaled width $d$ of the membrane by simple operation: $\mathrm{h[i]=d*h[i]}$. All grids are symmetric with respect to the center of the membrane.

There are several options concerning grid generation (the same for all three functions) and the prototype of each function is the same:

```
hGridGen(n, m, a, h[n], type)
```

where: *n* – the number of intervals, *a* – the fraction of the interval (near the boundaries or in the middle) that contains *m* intervals of equal length (see Fig. 1 – for the case of denser grid near boundaries, that is *type*=0, and Fig. 2 for the grid denser in the middle, that is *type*=1), *m* – the number of equidistance intervals (at the boundaries or in the middle), *type* – which case of the grid: 0 – for denser at the boundaries, 1 – for denser in the center of a membrane.

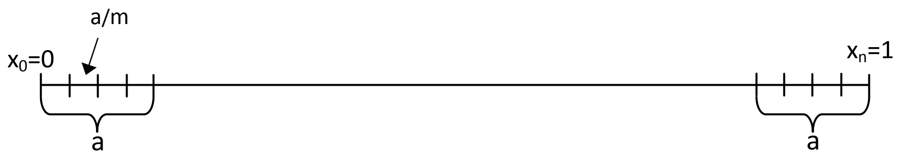

**Fig. 1. Definitions of the parameters (*a*) and (*m*) in a grid type=0. Type 0 means denser nodes distribution at the boundary region of the membrane.**

In standard simulations involving transport across the membrane interfaces (boundaries) the grid parameter $a$ is of order $10^{-4} \div 10^{-5}$. For $n=100$ a good choice of $m$ is in the range $10 \div 20$.

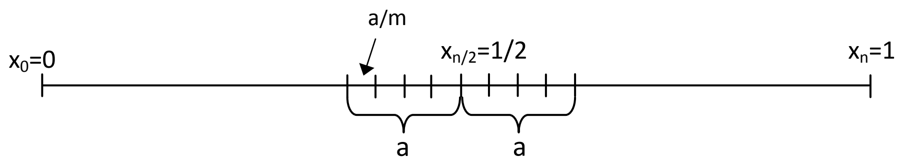

**Fig. 2. Definitions of the parameters (*a*) and (*m*) in a grid type=1. Type 1 means denser nodes distribution in the middle of the membrane.**

## Initial conditions

As was mentioned earlier the possible initial conditions are piece-wise constant according to the formula (1.1) for concentrations. Electric field is assumed to be zero at time $t=0$:

$$E(x,0)=0 \quad \Rightarrow \; E^0=0, E^1=0,\ldots,E^{n+1}=0, \tag{0.2}$$

what is written in the source code as

```
for(k=0; k <= n+1; k++) y[l(r,k)] = 0.0;
```

(remember that in the program we count species from zero, so for concentrations $i=0,\ldots,r-1$ and for electric field $i=r$). This type of initial conditions will later have to be changed for electrochemical impedance spectra (EIS) because obtaining them requires to simulate potential in time from the perturbed steady state system. This steady state usually will not be described by the step function concentration profiles (1.1) and (1.2). One of your task in the project will be to modify this part so as any initial distribution of concentrations $c_1(x,0),\ldots,c_r(x,0)$ and electric field $E(x,0)$ will be possible.