# C1 - Assignment 3 Report: Scalar initial value problems.

Student Number: 1894945

November 23, 2018



## Contents

# 1 Introduction

## 1.1 Initial value problem

# 2 The program

## 2.1 Implementation

The program is built on the class ADR , which stands for Advection-Diffusion-Reaction. The class has 7 private variables: $J$ (the number of points in the interior of $[0, L]$), $\alpha$, $\beta$, $\gamma$ (the parameters associated to the ADR equation at hand), $L$ (the length of the interval), $u_0$ (the boundary condition at 0) and $u_L$ (boundary condition at $L$). In this way, once the constructor is called, all the variables and parameters defining problem (**??**) are set.
As pointed out in §**??**, BVPs can be solved through numerical methods for linear algebra: the member function *MatrixBuild*, which accepts no argument, constructs matrix $A$ of Eqn.(**??**), which will be later inverted through the Gauss-Seidel algorithm. Such a construction takes place within the function *Solver*, that performs the tests required by the assignment instructions through the Gauss-Seidel method and outputs the results on different .txt files. For clarity reasons, each time a simulation is performed, a suitable success message containing the different parameters of the corresponding test, is printed on the terminal. *Solver* is a member function which accepts 4 arguments and implicitly returns the discretised solution *u_x*. The argument *itCheck* represents the number of iterations after which a check for stagnation of the method is performed, while *MaxIter* represents the maximum number of iterations the user is willing "wait" and *tol* the given tolerance.
The member function *An_sol* outputs the analytical solutions at each point of $\Omega_h$, by simply computing either Eqn.(**??**) or Eqn.(**??**) at the given set of points. The choice between the two solutions is made by means of an *if* instruction, checking if $\alpha \neq 0, \gamma = 0$ or if $\alpha \neq 0, \beta = 0$. The special case $\alpha \neq 0, \beta = 0, \gamma = 0$ is checked as well. If none of the above conditions is satisfied, the program exits with an appropriate error message to the user.
Discretised and the analytical solutions are finally compared within the function *ADR_Test*. Such a function takes as arguments the values of the private variables, the arguments of *Solver* and the different values of $J$ the user wants to test on. The function loops over them and prints the final error between the numerical solution and the analytical one, in the (discrete) $L_\infty$ norm, on suitable .txt files, whose names are defined through the parameters the user has set for the test. The logarithm of the ratio between errors obtained for different mesh sizes is computed and printed, too. Since, by means of simply changing the values provided in the main file, both class of equations can be tested with no further modifications to the code, the program can be regarded as particularly flexible.

## 2.2 Running the program

The program is run through the Makefile provided. The current optimisation is set to -Ofast and all following tests have been performed with this optimisation choice. Every possible error or warning has been explicitly checked by means of the instruction *-Wall -Wfatal-errors -pedantic* before proceeding to performing the different tests: no warnings appeared.
The program generates a fixed number of output files, in which the results of the tests have been stored. Some of these files are used to produce plots by means of six different plotscripts.

### 2.3 Testing correctness

In order to check the correctness of the implementation of the code, the tests requested by the assignment have been performed. The class of ADR equations chosen as a benchmark is the following:

$$-\alpha u''(x) + \beta u'(x) = 0 \tag{1}$$

Nonetheless, the correctness of the code for the other class of equations is easy to verify by means of changing the parameters in the main file. Such a test, whose results will not be reported here, has been successfully performed in the construction of the code.
The solution to Eqn.(1) has been discussed in detail in §**??**. However, from the numerical point of view, it is important to point out how BCs imply suitable modifications of the vector $\mathbf{f}$ to invert against. In fact, if the matrix $A$ is constructed to possess $J$ columns and $J$ rows, since the discrete Laplacian and gradient operators take as arguments $x_{j+1}$ and $x_{j-1}$ for the approximation at $x_j$, the values of the function $u(x)$ at the boundary of $\Omega_h$ have to be inserted into the first and last entry of the vector $\mathbf{f}$. Such a construction is performed in the function *Solver* trough the following lines:

```
f[0] = u0_*( ( alpha_/(h*h) ) + ( beta_/(2*h) ) ); //first boundary condition
f[J_-1] = uL_*( ( alpha_/(h*h) ) - ( beta_/(2*h) ) ); //second boundary condition
```

Eqn.(**??**) then reads:

$$\begin{pmatrix} \frac{2\alpha}{h^2}+\gamma & -\frac{\alpha}{h^2}-\frac{\beta}{2h} & 0 & 0 & 0 & \cdots & 0 & 0 \\ -\frac{\alpha}{h^2}+\frac{\beta}{2h} & \frac{2\alpha}{h^2}+\gamma & -\frac{\alpha}{h^2}-\frac{\beta}{2h} & 0 & 0 & \cdots & 0 & 0 \\ 0 & -\frac{\alpha}{h^2}+\frac{\beta}{2h} & \frac{2\alpha}{h^2}+\gamma & -\frac{\alpha}{h^2}-\frac{\beta}{2h} & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \cdots & \vdots & \vdots \\ 0 & 0 & \cdots & \cdots & \cdots & \cdots & \frac{2\alpha}{h^2}+\gamma & -\frac{\alpha}{h^2}-\frac{\beta}{2h} \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ \\ \vdots \\ u_{J-1} \end{pmatrix} = \begin{pmatrix} u_0\left(-\frac{\alpha}{h^2}+\frac{\beta}{2h}\right) \\ 0 \\ \vdots \\ 0 \\ u_L\left(-\frac{\alpha}{h^2}-\frac{\beta}{2h}\right) \end{pmatrix}$$

where $\mathbf{u}$ is the initial guess required by the Gauss-Seidel algorithm to work. In the present case, each component of $\mathbf{u}$ has been set to 0, so that $u_j = 0 \quad \forall j \in \{1, \cdots, J\}$.
Solutions to Eqn.(1) have been obtained for different values of the mesh size $h = \frac{1}{J+1}$ and different values of the $\mathbb{P}_e$. The error, i.e the distance between the exact solution on the grid and the approximate one in the (discrete) $L_\infty$ norm, is computed for each solution. As previously mentioned, the logarithm of the ratio between such errors for two successive solutions of the same problem (same $\mathbb{P}_e$, but different $h$) have been computed: the reason for this will be presented in §2.5.

### 2.4 Results of the tests

In the following, the results of the more significant tests performed will be reported and briefly discussed. The tolerance has been set to $10^{-6}$ for each of the tests, while the values of *itCheck* and *MaxIter* are kept fixed to $10^4$ and $10^9$ respectively.

### 2.4.1 Solutions to the AD equation

The *ADR_test* function has been shown to work correctly for the following Péclet numbers:

$$\mathbb{P}_e = \{0.0000, 0.0005, 0.0100, 1.0000, 10.5000\}$$

The solutions *u_x* obtained for the different cases, indexed by the respective $J$, can be found in the files *P_numb_..._Solution_J_....txt* , where the dots stand for the different possible values of $\mathbb{P}_e$ and $J$. The residual error at every iteration, in the $L_\infty$ norm, is stored in the files *P_numb_..._Residual_J_....txt*, which provide a direct proof of the general correctness of the code.
Finally, the distance between the exact solution and the approximate one, in the (discrete) $L_\infty$ is printed on *Errors_P_numb_...txt* files. Unsurprisingly, the error becomes smaller as the mesh size decreases.
The correctness of the algorithm has been tested for different values of $h$, as required by the assignment. The values of the mesh size have been chosen to be:

$$h = \{10, 20, 40, 80, 160, 320, 640\} \tag{2}$$

Keeping the Péclet number fixed, solutions have been obtained for all the previous mesh sizes.

### 2.4.2 Small Péclet numbers

Solutions to to Eqn.(1) have been studied in the case of small $\mathbb{P}_e$, as required by the assignment. The obtained results are reported in the following plots.
The solutions appear to be straight lines on the interval $[0, 1]$. Remembering what has been discussed in §**??**, such a result does not appear surprising: Fig.**??** and Fig.**??** are in good agreement with the Taylor expansion performed in Eqn.(**??**). Finally, Fig.**??** shows that the code is able to find the correct solution to the problem also in the case of $\beta = 0$. In fact, under these circumstances, Eqn.(1) simplifies to:

$$-\alpha u''(x) = 0$$

which, for the given BCs, admits the unique solution $u(x) = x$.

### 2.4.3 Péclet number close or equal to 1

Solutions to Eqn.(1) have been studied in the case $\alpha = 0.25, \beta = 0.5$ as well. Such a choice of the parameters results in $\mathbb{P}_e = 1$. The result is reported in the following plot.
The exponential behaviour of the solution is now evident. In fact, in the present case, the Taylor expansion of the exponential function around 0 cannot be truncated after the first term being that $\frac{\beta}{\alpha} = 1$.
Solutions overlap in this case as well, but small differences between them are noticeable for big enough $j$: by means of a suitable zoom on the plot, it is possible to see that the red and blue curve present a slightly different profile.
Fig.**??** indirectly provides a proof of the correctness of the analysis performed in §**??**. Not only in the present situation it is not possible to perform an expansion about 0 of the exponential, but also

it is not possible to use the approximation performed for large $\mathbb{P}_e$, as confirmed by the absence of boundary layers in the plot.

### 2.4.4 Large Péclet number

Finally, solutions to Eqn.(1) have been studied in the case of large $\mathbb{P}_e$. The results are reported in the following plot.
The presence of a wide boundary layer is noticeable: all solutions stay equal to naught for $0 \leq x < 0.8$ and then reach 1.

The choice of the values of $\alpha$ and $\beta$ has been made through the following criterion.
A sufficient condition for the Gauss-Seidel method to converge is that the matrix $A$ which the algorithm has to invert is strictly diagonally dominant (see [1]). In the present case, such a condition is satisfied if:

$$\left\| \frac{2\alpha}{h^2} \right\| > \left\| \frac{\alpha}{h^2} + \frac{\beta}{2h} \right\| + \left\| -\frac{\alpha}{h^2} + \frac{\beta}{2h} \right\|$$

However:

$$\left\| \frac{\beta}{h} \right\| = \left\| -\frac{\alpha}{h^2} + \frac{\beta}{2h} + \frac{\alpha}{h^2} + \frac{\beta}{2h} \right\|$$

So that, in order to have strict diagonally dominance, it must be:

$$\left\| \frac{\beta}{h} \right\| < \left\| \frac{2\alpha}{h^2} \right\|$$

Hence, for $L = 1$, remembering that $h = \frac{1}{J+1}$, one has that a sufficient condition for the Gauss-Seidel method to converge is:

$$J > \mathbb{P}_e - 1$$

The choice of $\alpha = 1.0$ and $\beta = 21.0$ satisfy the above condition. It was explicitly checked during the construction of the program that the choice of a great $\beta$ (e.g. $\beta = 1000$) does not allow the algorithm to converge and the program exits with an appropriate warning to the user. Clearly, large values of $\beta$ can be tested with $J$ large enough to make the previous inequality satisfied.

## 2.5 Error analysis and order of convergence

As required by the assignment, solutions errors have been computed and analysed. Not surprisingly, the distance between analytical and the approximate solution becomes smaller when the mesh size decreases. The study of such errors becomes of great importance to determine the order of convergence of the method. Assume the FD method here used is convergent of order p, then:

$$\left\| \frac{r_h u - u}{r_{\frac{h}{2}} u - u} \right\|_{\bar{\mathbb{U}}_h} = \frac{Ch^p + \mathcal{O}(h^{p+1})}{C\left(\frac{h}{2}\right)^p + \mathcal{O}(h^{p+1})} = 2^p + \mathcal{O}(h)$$

Passing to the logarithms:

$$\log_2 \left\| \frac{r_h u - u}{r_{\frac{h}{2}} u - u} \right\|_{\bar{\mathbb{U}}_h} = p + \mathcal{O}(h) \tag{3}$$

The order of convergence of a method can thus be evaluated by means of studying the behaviour of the distance in the $\bar{\mathbb{U}}_h$ norm between the discretised and the exact solution to a given problem.
The choice of progressively reducing by a factor 2 the values of $h$ (see Eqn.(2)) should now appear clear: such a choice guarantees that the data are uniformly distributed on a logarithmic scale, which, in turn, makes the analysis of the errors simpler.
As already mentioned, in the present case the (discrete) $L_\infty$ (see [2]) norm has been chosen.

Fig.**??** shows a clear dependence of the error empirical profiles on the Péclet number. In particular, it is apparent that when $\mathbb{P}_e << 1$ the error scales approximatively uniformly on a logarithmic scale with respect to $h$.
For $\mathbb{P}_e \geq 1$ the empirical profiles become very similar to the analytical $h^2$ one, suggesting that the order of convergence of the method is equal to 2. The curve obtained in the case of $\mathbb{P}_e = 1$ shows another interesting feature, i.e. the presence of a *plateau* for small $h$ values. This is, however, is not surprising: the error cannot become arbitrarly small when the mesh size decreases and a saturation of the performances of the method for small enough $h$ is expected. As a matter of fact, a saturation for large values of the mesh size is expected as well: such a region, however, is not extremely interesting from a numerical point of view (it is just signalling that the choice of the mesh size has not been made optimally) and has not been investigated.
A more precise analysis can be performed through the data contained in the file *Errors_P_numb_10.500000.txt*, which is reported in the following:

```
Details of numerical method for ADR equation through Gauss-Seidel algorithm for alpha=1,
    beta=21, gamma=0.
#1-Mesh size          2-Error (LinfNorm)     3-log2(Error Ratios)
0.1                   0.146847
0.05                  0.0384623              1.93279
0.025                 0.0086967              2.14491
0.0125                0.00212548             2.03268
0.00625               0.000528445            2.00797
0.003125              0.00013208             2.00034
0.0015625             3.30134e-05            2.00029
```

The third column of the file contains the results of Eqn.(3) for $\mathbb{P}_e = 10.500$. It is evident that the method has indeed order of convergence 2.
By means of computing the empirical mean and standard error of the numerical results obtained, one has:

$$O_c = 2.02 \pm 0.07 \tag{4}$$

where $O_c$ represents the order of convergence of the method.
The result is justifiable through Thm.**??**, which, as already pointed out, shows that, for stable methods, the order of convergence is at least the order of consistency of the method.

# 3 Conclusive remarks

In this section some additional comments and observation are presented.

## 3.1 Memory

As for the previous assignment, possible memory leaks have been checked.with the help of the software *valgrind*, To do this, it is necessary to compile with *-g* and *-O1* optimisation. After having compiled the program, the following instruction has been used:

```
$ valgrind --leak-check=yes ./adr
```

The following output has been produced on the terminal:

```
==31379== Memcheck, a memory error detector
==31379== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==31379== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==31379== Command: ./adr
==31379==
==31379== HEAP SUMMARY:
==31379==     in use at exit: 0 bytes in 0 blocks
==31379==   total heap usage: 244,730 allocs, 244,730 frees, 130,033,803 bytes allocated
==31379==
==31379== All heap blocks were freed -- no leaks are possible
==31379==
==31379== For counts of detected and suppressed errors, rerun with: -v
==31379== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

One can clearly see that no memory has been lost. This is however not surprising since no memory was dynamically allocated by the program and the correctness of the constructor for the SparseMatrix class was checked in the previous assignment.

## 3.2 Performances

Performances can be checked through the following commands:

```
$ valgrind --tool=callgrind ./adr
$ kcachegrind
```

The first command analyses the perfomances in the terms of load distrubution, while *kcachegrind* is used to visualise the load distribution results. Differently from the previous case, performances depend on the code of the first assignment. The functions called within *ADR_Test* and *Solver* are the ones taking up the majority of the time needed: unsurprisingly, *Gauss-Seidel* is the most "time-consuming" of them.

## References

[1] A. Quateroni, R. Sacco, F. Saleri; *Numerical Mathematics*, Vol.37, Springer Verlag, (2007).

[2] T. Grafke; *Scientific Computing*, Lecture Notes, University of Warwick, (2018).