C1 - Assignment 4 Report: Parabolic Initial-Boundary Value Problem

Student Number: 1894945

December 21, 2018

Contents

1 Introduction

Intuitively, (Parabolic) Initial-Boundary Value Problems (IVBPs) combine an initial value problem in a distinguished "time-like" direction with a boundary value problem in "space-like" directions. In this assignment the approximation of solutions to them for scalar parabolic differential equations (PDEs) will be considered.

Definition 1. Let $I \subset \mathbb{R}$ be an interval, $\Omega \subset \mathbb{R}$ be a simply connected, bounded domain $t_0 \in I$, $\mathcal{L}$ a spatial differential operator of the form $\mathcal{L} = -\Delta + \mathbf{p} \cdot \nabla + q$, for $\mathbf{p}$ and q smooth. The parabolic initial boundary-value problem (IBVP) is to find $u(\mathbf{x}, t) : \Omega \times I \to \mathbb{R}$ for which:

$$\begin{cases} \partial_t u + \mathcal{L}u = f \quad \text{for } \mathbf{x} \in \Omega, \ \ \forall t \in \ I \\ u = v \quad \text{for } \mathbf{x} \in \partial\Omega, \ \ \forall t \in I \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}) \quad \text{for } \mathbf{x} \in \Omega \end{cases} \tag{1}$$

with data $f(\mathbf{x}, t) : \Omega \times I \to \mathbb{R}$, $v(\mathbf{x}, t) : \partial\Omega \times I \to \mathbb{R}$, $u_0(\mathbf{x}) : \Omega \to \mathbb{R}$. Eqn.(1) is a parabolic partial differential equation (PDE).

In general, in the spatial direction, one wants the operator $\mathcal{L}$ to be uniformly elliptic in order for the whole problem to be parabolic .

While well posedness for ODEs is satisfactorily answered in a generic way with the Picard-Lindelöf theorem, the same is not true for PDEs, not even linear ones. Often, existence and uniqueness are established on a case-by-case basis, and in many cases is not known at all (for example famously for the Navier-Stokes equation describing the motion of incompressible fluids). Nevertheless, there is huge interest from an applications perspective to obtain numerical solutions to PDE problems relevant to physics and engineering. General PDE theory will not be presented here. Instead, methods to combine known methods for IVPs and BVPs to solve problems of the type of problem 1 will be discussed. Clearly, the idea is to discretise the spatial direction by replacing $\mathcal{L}$ with an appropriate $\mathcal{L}_h$, and additionally using one of the time integration schemes introduced above to treat the time evolution.

The following simple problem represents the benchmark problem for testing numerical schemes for IVBP in the present assignment.

Definition 2. Let $I = [0, T]$, $\Omega = (0, L)$, where $L \in \mathbb{R}^+$ and $T \in \mathbb{R}^+$, and consider the following IVBP:

$$\begin{cases} \partial_t u - k\partial_x^2 u = f \quad \text{for } x \in \Omega, \ \ \forall t \in \ I \\ u(0, t) = u(1, t) = 0 \quad \text{for } x \in \partial\Omega, \ \ \forall t \in I \\ u(x, 0) = u_0(x) \quad \text{for } x \in \Omega \end{cases} \tag{2}$$

Eqn.(2) is called the heat equation. Specifically, $u(x, t)$ describes the diffusive transport (of matter or heat) in space, with constant diffusivity/conductivity $k \in \mathbb{R}^+$, and where $f(x, t)$ describes the local heat production. The heat equation is well posed.

1.1 Numerical methods to approximate solution to IBVPs

Two different approaches to finding numerical solution to IBVPs can be adopted. The first one results in the so-called *semi-discretisation schemes*, in which a generic IBVP is first discretised in space and then in time or viceversa. Alternatively, a simultaneous discretisation, leading to a *fully discretised scheme*, can be performed.
A detailed presentation of *fully-discretised schemes*, which not be covered here, can be found in Ref.[1]. In the present report it will only be pointed out that, although a fully-discretised approach appears to be more convenient, in the case of stiff problems approximated by means of method characterised by a bounded region of absolute stability, the choice of the discretisation time step and of the mesh size cannot be done arbitrarily, since the meshsize and the time step are related by the Courant-Friedrichs-Levy conditions (see Ref.[1]).

1.1.1 Methods of lines

In the present assignment only one of two possible approaches to semi-discretistion will be presented. A more general discussion of them can again be found in Ref.[1] and Ref.[2].
In particular, only the so called *method of lines* will be presented.

The method of lines takes the position of first discretising in space. The resulting problem becomes a system of ordinary differential equations. The semidiscretisation generates an IVP for a system of ODEs which may be solved by known methods for IVPs. To discretise the spatial derivatives one shall apply the classical central difference scheme for the Laplacian operator, with an equidistant spatial grid. Applying this to Eqn.(2), the following system of ODEs is obtained:

$$\begin{cases} \frac{du_i}{dt} = \partial^+\partial^- u_i + f_i \quad \forall x_i \in \Omega_h \\ u_i = 0 \quad \forall x_i \in \Gamma_h \end{cases} \tag{3}$$

with initial conditions $u_i(0) = u_0(x_i)$ for $x_i \in \Omega_h$.
The notation is the same used in Assignment 2 and Assignment 3 report, i.e. the notation of Ref.[2].
To complete the discretisation a solver for ODEs has to be applied to the system.

1.2 The present case

In the present assignment Eqn.(2) for $L = 1$ and $T = 0.1$ and periodic boundary conditions, will be solved through the system of ODEs (2) by means of RK methods. In particular, the forward Euler method, the backward Euler method and the 3-stage Heun method will be used.
As for the initial condition, the following is required:

$$u_0(x) = \begin{cases} 0 & \text{if } x \leq \frac{1}{4},\ x > \frac{3}{4} \\ 1 & \text{otherwise} \end{cases} \tag{4}$$

Finally, the datum f is set to 0.

2 The program

2.1 Implementation

The program is built on the classes and struct that can be found in the files *schemes.hh* and *models.hh*, which contain the different RK methods to use and the different benchmark problem to test on, respectively.

2.1.1 schemes.hh file

The class DIRK is made of 4 *protected* variables: the matrix A, the vector b, the vector c and the integer s (named stages_ in the code), which fully characterise the Butcher tableau of any RK method. This is done through the following instructions, which can be found in the schemes.hh file:

```
protected:

   int stages_;

   std::vector<double> a_,b_,c_;
```

It may be worth pointing out that, even though $a\_$ represents a matrix, it is implemented as a vector in the program.
The classes FE, BE, IM, $Heun3$ and $DIRK2$ are implemented as derived classes of $DIRK$. Their names stand respectively for: Forward Euler, Backward Euler, Implicit Midpoint, 3-stage Heun and 2-stage DIRK. The Butcher tableau of the methods can be found in the assignment instruction and will not be reported here.

The function *evolve*, which returns the evolved approximated solution u_{n+1}, u_n being the present approximation, can be found in the file as well. For flexibility reasons, the function is built as a *template*: in this way, both methods can be accessed by the function. The prototype of the routine, is displayed in the following:

```
double evolve(const double &y,double t,double h,const Model &model,unsigned &count)
    const //one step forward
```

Here y represents the approximated solution at the time t, with h the chosen discretisation stepsize. In case an implicit RK method is chosen, Eqn.**??** becomes an implicit equations for K_i and has to be solved by means of numerical methods. As required by the assignment instructions, Newton's method has been used.

```
if( a(i,i)!=0 ) //implementing Newton's method
{
   unsigned iter = 0;
   double error = ( -k[i] + model.f(t + c_[i]*h, tmp_sum + h*a(i,i)*k[i]) );
```

```
count ++;
double den; //den stays for the denumerator of the fraction which updates k[i] in
    Newton's method
while(iter < 1e6 && std::abs(error) > 1e-6 ) //1e6 is maximum number of iterations
    and 1e-6 is the given tol
{
   den = -1 + model.df( t + c_[i]*h, tmp_sum + k[i] )*h*a(i,i);
   count++;
   k[i] -= error/den;
   error = -k[i] + model.f(t + c_[i]*h, tmp_sum + h*a(i,i)*k[i]) ;
   count++;
iter ++;
}
```

As it can be seen from the listed code above, the program checks if Eqn.(**??**) is an implicit equation for K_i: if the entry $a_{ii} \neq 0$, then Newton's method is implemented. The error at every iteration, which is equal to $f\left(t + hc_i, u_n + h\sum_{j=1}^{i-1} a_{ij}K_j + ha_{ii}K_i\right)$, where f is the one defined in Eqn.(**??**), is contained in the variable *error* and happens to be equal to the numerator of the fraction *error/den* which updates the value of K_i ($k\_[i]$ in the code) at every iteration. For this reason, the same variable has been used to both store the error and construct the updating of K_i, making the routine more efficient. The initial guess for the solution has always been chosen to be equal to naught.
The variable *tol* represent the tolerance and has been here set to 10^{-6}: the method stops once the distance (in L_∞ norm) between the RHS and LHS of Eqn.(**??**) is smaller than the fixed tolerance or after a fixed number of iterations (which has been chosen to be equal to 10^6).
Finally, *count* represents the number of evaluations of f and the derivatives of f (see §2.1.2). Such an evaluation is required by the assignment instructions and allows to perform a basic efficiency analysis of the different methods employed.

2.1.2 models.hh file

In this file a *struct*, whose elements completely characterise the model at hand, is defined.

```
struct Test
{
   unsigned modelNumber_;
```

The stuct contains the variable *modelNumber_*, which allows the user to chosse between different models to test on. Such models are indexed by means of the values $1, 2$ and 3, with an obvious correspondence with the assignment instructions. The choice of *modelNumber_* 3, is associated with problem (**??**).
The other elements of *Test* are f, df, T and y_0 and *exact*, which represent the function f defined in each model, its derivative, the total time of integration, the starting point and the exact solution respectively. It should now be clear what the instructions *model.f* and *model.df* listed in §2.1.1 stands for: they allow the user to compute f and its first derivative at the given point (which is given as an argument to model).
The choice between the different models is made by means of different *if* instructions. As an

example, one of them is reported in the following.

```
double f(double t,const double &y) const
{
if(modelNumber_==1) //first assignment equation
{
   return (1 - cos(t)*(cos(t) - 1) - y*y);
}else if(modelNumber_==2) //second assignment equation
{
   double l = -0.1;
   return (1 - cos(t)*(cos(t) - exp(l*t)) - exp(-2*l*t)*y*y + l*y);
}else if(modelNumber_==3) //test case
{
   return -y;
}else
{ //exit with an appropriate warning
   std::cout << "No model defined for this choice of model index. Exiting." << std::endl;
   exit(EXIT_FAILURE);
}
```

The above listed code defines the form of the function $f(y,t)$, i.e. the models the user can test on. It is worth pointing out that, in the event of a non correct choice of the model, the program exits with an appropriate error to the user.

2.1.3 main.cc file

The main file contains the function *solve* which performs the integration of the chosen equation up to T. The function contains an *if* instruction, which enables the printing of the approximation error over time for two different step sizes, which have been chosen to be $\tau_0 = 0.1$ and $\tau_0 = 0.05$, for the first model. The files are named by means of the following instruction:

```
if( ( tau==0.1 || tau==0.05) && (model.modelNumber_ == 1) )
{
   std::ofstream myoutFile;
   myoutFile.open("Error_Model_" + std::to_string(model.modelNumber_) + "_Scheme_" +
       std::to_string(schemeNumb) + "_tau_" + std::to_string(tau) + ".dat",
       std::ios::out);
```

The integration up to the time T is performed by means of repeated calls to *evolve*.

```
while ( t<=model.T() ) //up to T
{
   y =  scheme.evolve(y,t,tau,model, count); //one step
   t += tau; //increment in time
   double error = y - model.exact(t);
```

```
   maxError = std::max(maxError,std::abs(error));
}
```

In case $\tau_0 \neq \{0.1, 0.05\}$ and the chosen model is not the first one, the function returns the maximum error in the L_∞ norm between the exact solution and the approximated one over the whole integration time T, as the above listed code shows.

The program accepts 4 arguments from the prompt (here the Makefile) and, whenever an incorrect assertion of them happens, exits with an appropriate warning the user. A vector of pointers to DIRK is constructed.

```
std::vector<DIRK *> schemes;
schemes.push_back(new FE());
schemes.push_back(new BE());
schemes.push_back(new IM());
schemes.push_back(new Heun3());
schemes.push_back(new DIRK2());
```

By accessing a suitable location of this vector, each of the RK methods can be employed. The *experimental order of convergence* (eoc) for each time step is printed on suitable files, named through the following instruction:

```
std::ofstream myFile;
myFile.open("EOCS_Model_" + std::to_string(modelNumber) + "_Scheme_" +
    std::to_string(schemeNumber) + "_tau_" + std::to_string(tau) + "_J_" +
    std::to_string(level) + ".dat", std::ios::out);
if(!myFile.good())
{
   std::cout << "Failed to open the file." << std::endl;
   exit(EXIT_FAILURE);
}
```

As for the previous cases of file-opening instructions, if the program cannot open the file, an appropriate warning message is printed on the terminal. All the files have been made human readable for easing the reading of them by generic users.
Finally, the number of evaluations of f and df (summed over the whole sequence of stepsizes, so that result is independent of the specific stepsize chosen) for each method is printed on the terminal every time the integration is concluded. A suitable message informing the user that the integration has ended is printed as well.

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 four 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: tests on model 1 and 2 have been performed. 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.

2.4 Results of the tests

In the following, the results of the more significant tests performed will be reported and briefly discussed.

2.4.1 Experimental errors of convergence

The experimental error of convergence for each combination of models, schemes and stepsizes[1], has been computed. As shown in the previous section, each scheme is indexed though the number of the location it occupies in the vector of pointers to DIRK. In particular, 0 is associated with FE, 1 is associated with BE, 2 is associated with BE, 3 is associated with IM, 4 is associated with Heun3 and 5 is associated with DIRK2.

The results are contained in the "Errors" files, whose content is reported in the following:

```
#EOCS for scheme 0, relative to model 1
#1-tau                2-Maximum Error       3-EOCS
0.1                   0.126438
0.05                  0.035277              1.84163
0.025                 0.00949667            1.89323
0.0125                0.00304776            1.63967
0.00625               0.00152847            0.995661
0.003125              0.000765525           0.997569
0.0015625             0.000383107           0.9987
0.00078125            0.000191643           0.999326
0.000390625           9.58445e-05           0.999656
0.000195313           4.7928e-05            0.999826
9.76563e-05           2.39654e-05           0.999913
4.88281e-05           1.19831e-05           0.999956
2.44141e-05           5.99164e-06           0.999978
```

[1] In the following, the stepsizes will be denoted both with h, in accordance to the above theoretical sections, and with τ, in accordance with the assignment instuctions.

```
#EOCS for scheme 0, relative to model 2
#1-tau                 2-Maximum Error       3-EOCS
0.1                    inf
0.05                   inf                   -nan
0.025                  inf                   -nan
0.0125                 inf                   -nan
0.00625                0.481524              inf
0.003125               0.132811              1.85823
0.0015625              0.057307              1.2126
0.00078125             0.0269536             1.08823
0.000390625            0.013101              1.0408
0.000195313            0.0064618             1.01967
9.76563e-05            0.00320933            1.00967
4.88281e-05            0.00159934            1.00479
2.44141e-05            0.000798349           1.00239
```

Regarding the FE method, it is clear that the order of convergence of the method is one. This is in perfect agreement with the theoretical predictions. As a matter of fact, Thm.**??** guarantees that the order of consistency of the method is one: being that the method is zero-stable, its order of convergence has to be at least equal to the order of consistency.
It is worth pointing out that the method appears to be non-stable (in the sense of zero stability) for time steps which are bigger than 0.00625. This is again in perfect agreement with Def.**??**, which defines stability for times steps smaller or equal of a fixed h_0: the results suggest that h_0 for model 2 and the FE method is smaller than 0.0125.

For the BE scheme, one has:

```
#EOCS for scheme 1, relative to model 1
#1-tau                 2-Maximum Error       3-EOCS
0.1                    0.39611
0.05                   0.055257              2.84167
0.025                  0.0117679             2.2313
0.0125                 0.00310423            1.92255
0.00625                0.00154073            1.01062
0.003125               0.000768528           1.00345
0.0015625              0.000383855           1.00154
0.00078125             0.00019183            1.00073
0.000390625            9.58911e-05           1.00036
0.000195313            4.79397e-05           1.00018
9.76563e-05            2.39684e-05           1.00009
4.88281e-05            1.19838e-05           1.00004
2.44141e-05            5.99181e-06           1.00002
```

```
#EOCS for scheme 1, relative to model 2
#1-tau                 2-Maximum Error       3-EOCS
```

```
0.1                 0.639798
0.05                0.495569            0.368531
0.025               0.358309            0.467881
0.0125              0.239848            0.579083
0.00625             0.148207            0.694511
0.003125            0.0851809           0.79901
0.0015625           0.0462974           0.8796
0.00078125          0.0242502           0.93293
0.000390625         0.0124282           0.964386
0.000195313         0.00629377          0.981617
9.76563e-05         0.00316733          0.990657
4.88281e-05         0.00158885          0.995288
2.44141e-05         0.000795727         0.997634
```

For the same reasons of the previous case, one can argue that the order of convergence is one and that such a result is in agreement with what one expects theoretically.

In the case of the BE method, Thm.**??** guarantees that its order of consistency is 2 ($bc = 0.5$, $b = 1$). Again, making use of its stability, one finds the experimental data to be in agreement with the theoretical predictions.

```
#EOCS for scheme 2, relative to model 1
#1-tau              2-Maximum Error     3-EOCS
0.1                 0.0585369
0.05                0.0151887           1.94635
0.025               0.00383395          1.9861
0.0125              0.00096082          1.99649
0.00625             0.000240347         1.99915
0.003125            6.00952e-05         1.9998
0.0015625           1.50244e-05         1.99995
0.00078125          3.75613e-06         1.99999
0.000390625         9.39042e-07         1.99999
0.000195313         2.3469e-07          2.00044
9.76563e-05         5.86338e-08         2.00095
4.88281e-05         1.49431e-08         1.97226
2.44141e-05         3.88959e-09         1.94179
```

```
#EOCS for scheme 2, relative to model 2
#1-tau              2-Maximum Error     3-EOCS
0.1                 0.209799
0.05                0.072592            1.53112
0.025               0.0203654           1.83369
0.0125              0.00526223          1.95237
0.00625             0.00132671          1.98782
0.003125            0.000332426         1.99675
0.0015625           8.31018e-05         2.00008
0.00078125          2.07828e-05         1.99949
```

```
0.000390625             5.19649e-06             1.99978
0.000195313             1.29881e-06             2.00035
9.76563e-05             3.24474e-07             2.00101
4.88281e-05             8.2711e-08              1.97195
2.44141e-05             2.16242e-08             1.93543
```

Regarding the Heun3 scheme, following the same line of reasoning of the previous cases, it is possible to argue that, in agreement with the theoretical predictions, the order of consistency is equal to 3.

```
#EOCS for scheme 3, relative to model 1
#1-tau                  2-Maximum Error         3-EOCS
0.1                     0.000312788
0.05                    1.96209e-05             3.99473
0.025                   1.22761e-06             3.99847
0.0125                  8.95918e-08             3.77634
0.00625                 1.10018e-08             3.02563
0.003125                1.36331e-09             3.01255
0.0015625               1.69549e-10             3.00733
0.00078125              2.2073e-11              2.94135
0.000390625             1.39465e-11             0.662379
0.000195313             6.74928e-11             -2.27483
9.76563e-05             5.50014e-11             0.295266
4.88281e-05             2.70395e-10             -2.29753
2.44141e-05             2.21423e-10             0.288268
```

```
#EOCS for scheme 3, relative to model 2
#1-tau                  2-Maximum Error         3-EOCS
0.1                     0.00208592
0.05                    6.50329e-05             5.00337
0.025                   4.32878e-06             3.90914
0.0125                  1.31976e-06             1.71369
0.00625                 2.13607e-07             2.62724
0.003125                2.97629e-08             2.84337
0.0015625               3.9283e-09              2.92154
0.00078125              4.06131e-10             3.27389
0.000390625             1.98413e-11             4.35537
0.000195313             3.85861e-10             -4.28151
9.76563e-05             3.31035e-10             0.221099
4.88281e-05             1.50893e-09             -2.18847
2.44141e-05             1.31991e-09             0.193087
```

It is worth noticing that the eoc either gets close to zero or becomes negative for small enough step sizes: this behaviour is somewhat expected and signals a saturation of the method performances with the reduction of the step size.

Finally, the results for the DIRK2 method are presented.

```
#EOCS for scheme 3, relative to model 1
#1-tau                  2-Maximum Error        3-EOCS
0.1                     0.000312788
0.05                    1.96209e-05            3.99473
0.025                   1.22761e-06            3.99847
0.0125                  8.95918e-08            3.77634
0.00625                 1.10018e-08            3.02563
0.003125                1.36331e-09            3.01255
0.0015625               1.69549e-10            3.00733
0.00078125              2.2073e-11             2.94135
0.000390625             1.39465e-11            0.662379
0.000195313             6.74928e-11            -2.27483
9.76563e-05             5.50014e-11            0.295266
4.88281e-05             2.70395e-10            -2.29753
2.44141e-05             2.21423e-10            0.288268
```

```
#EOCS for scheme 3, relative to model 2
#1-tau                  2-Maximum Error        3-EOCS
#1-tau                  2-Maximum Error        3-EOCS
0.1                     0.00280051
0.05                    0.000764064            1.87393
0.025                   0.000154804            2.30325
0.0125                  2.26948e-05            2.77001
0.00625                 3.21506e-06            2.81945
0.003125                4.45826e-07            2.85029
0.0015625               1.01866e-07            2.1298
0.00078125              1.11012e-08            3.19789
0.000390625             1.31348e-09            3.07925
0.000195313             5.51304e-10            1.25248
9.76563e-05             3.51625e-10            0.648811
4.88281e-05             1.50797e-09            -2.1005
2.44141e-05             1.32134e-09            0.190605
```

Once again, making use of Thm.**??** and Thm.**??**, one has that the order of convergence has to be at least two. Being that maximum errors are closer to the ones of the Heun3 method, the order of convergence is expected to be 3, in agreement with theoretical predictions.

In order to have a simpler visualisation of the dependence of the maximum approximation error on the time step, the following plots have been produced. By means of plotting the theoretical profiles τ, τ^2 and τ^3 on the same plots, a direct confirmation of the previous results can be obtained.

The plots confirm what has been previously stated through Thm.**??** and Thm.**??**. As a matter of fact, the good agreement between the τ and the profiles obtained for the FE and BE methods is clearly observable in Fig.**??** and Fig.**??**. It is also clear that the convergence error of the IM method is equal to 2. Moreover, the plots confirm what has been said above about the order of convergence

of the DIRK2 method: its profile appears to be in good agreement with the theoretical profile τ^3 and with the Heun3 one. The aforementioned saturation of the performances of the DIRK2 and Heun3 method for small step sizes is clearly observable in both plots.

2.4.2 Approximation error over time

The approximation error, computed in the L_∞ norm, has been studied as a function of the integration time for model 1 for two different step sizes, i.e. $h = 0.1$ and $h = 0.05$. The results are reported in the two following plots:
Similar curves are observed in the plots. As expected, the error tends to grow with time, even if, for specific time intervals, a decreasing profile is present.

2.4.3 Efficiency of the methods

A basic analysis of the performance of each of the methods is performed by means computing the number of times f and df are computed for each method. The results are output on the terminal so that a generic user can easily verify them.
Unsurprisingly, the most performant method is the FE method. This can be understood by noticing that the method is an explicit one-stage method: usage of Newton's method is not required by the time evolution and the total number of evaluations of f and df is the minimum possible since no loops over s, s being the number of stages of the method, is performed.

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 ./myprogram 1 0 0.1 12
```

In the present case the parameters refer to the case of model 1, FE scheme, $\tau_0 = 0.1$ and $J = 12$.

The following output has been produced on the terminal:

```
==4559== Memcheck, a memory error detector
==4559== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==4559== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==4559== Command: ./myprogram 1 0 0.1 12
==4559==
```

```
Solved Model 1 with scheme 0
Number of calls to f function and df function : 514661
==4559==
==4559== HEAP SUMMARY:
==4559==     in use at exit: 0 bytes in 0 blocks
==4559==   total heap usage: 514,706 allocs, 514,706 frees, 4,221,808 bytes allocated
==4559==
==4559== All heap blocks were freed -- no leaks are possible
==4559==
==4559== For counts of detected and suppressed errors, rerun with: -v
==4559== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

One can clearly see that no memory has been lost. Hence, the instruction:

```
for (const auto& r : (schemes)) //freeing memory
{
  delete r;
}
```

which is contained in the main file, works correctly.

3.2 Performances

Performances can be checked through the following commands:

```
$ valgrind --tool=callgrind ./myprogram 1 0 0.1 12
$ kcachegrind
```

where the parameters inserted from terminal have to be changed according to the desired model, scheme, initial step size and number eocs computed. In the present case the parameters refer to the case of model 1, FE scheme, $\tau_0 = 0.1$ and $J = 12$.
The first command analyses the perfomances in the terms of load distrubution, while *kcachegrind* is used to visualise the load distribution results.
Unsurprisingly, the most "time-consuming" function of the code appears to be *solve*.

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).