PADOVA UNIVERSITY PRESS

Dolomites Research Notes on Approximation

Special Issue SA2022 – *Software for Approximation 2022*, Volume 15 · 2022 · Pages 47–65

A MATLAB code for the computational solution of a phase field model for pitting corrosion*

Dajana Conte[a] · Gianluca Frasca-Caccia[a]

Abstract

Phase field models have been widely considered to simulate corrosion dynamics characterised by moving boundaries. The benefits of using these models rely on the fact that the moving interface is implicitly treated by means of the introduction of an auxiliary variable. However, the computational cost of these methods is typically very high. In this paper we consider a model for pitting corrosion of a metallic specimen immersed in an electrolytic solution. For its numerical solution we consider a method that relies on a suitable splitting of the governing equations and on the use of exponential integrators. The use of modern MATLAB functions to evaluate the effect of matrix exponentials on a vector is crucial for the efficient implementation of the method. The software used is presented and discussed in detail, and some numerical tests are introduced to show the performance of the proposed algorithms.

1 Introduction

Mathematical modelling based on chemical reactions and physical laws, is crucial to understand the evolution of corrosion processes and to predict the damages that it may cause. The solution of these mathematical models typically requires the use and analysis of numerical methods, and their validation and calibration by means of both laboratory and computer simulations.

Corrosion phenomena are typically modelled by diffusion, reaction-diffusion or advection-diffusion Partial Differential Equations (PDEs) for which a wide range of numerical methods is available in the literature [2,4–6,9–14,16,20,21,24,26,40,41].

In this paper we consider a corrosion system composed by a metal, such as 304 stainless steel, immersed in an electrolyte solution, such as NaCl. Metal surfaces are typically covered by a protective film that prevents general corrosion. However, this film may occasionally break. In this case the metal enters in direct contact with the electrolyte and locally dissolves, leading to a phenomenon known as pitting corrosion.

The initiation of the process causing the film breakdown has a stochastic nature and occurs on very minute scales (on the order of nanometres) and in very short times (on the order of microseconds), so it is very hard to detect. Moreover, this process also greatly relies on the nature of the material surface and on the environmental parameters. For these reason the initiation phenomenon is not fully understood yet [1].

Empirical models, including stochastic, probabilistic and neural network models, have been proposed in the literature [1,28,30,31,36–38], to simulate all phases of pitting formation: initiation, nucleation and growth. These are typically modelled as Poisson, Weilbull and Markov processes, respectively [1,38]. Empirical models require large database for calibration which is unavailable on failure events. So it is hard to construct well-calibrated empirical models, and their predictive capability is generally limited [18].

We study here the diffusion of an existing pit simulated by the deterministic phase field model in [29]. Phase field models have been widely used to simulate moving interface problems in a variety of contexts (see, e.g., [3,8,25,34,39]).

We consider a spatial computational domain, $\Omega \subset \mathbb{R}^n$, where typically $n \in \{2,3\}$, containing both the metal and the solution phases. Let be c a normalized variable accounting for the molar concentration of metal atoms or ions at each point.

In a phase field model the interface evolution is implicitly taken into account by introducing an auxiliary phase field variable, ϕ. The variable ϕ has constant values $\phi = 0$ and $\phi = 1$ in the bulk of the electrolyte and the metal phases, respectively, and varies continuously across the interface between the two phases. Therefore, by definition,

$$(\phi(\mathbf{x},t), c(\mathbf{x},t)) \in [0,1]^2, \qquad \mathbf{x} \in \Omega, \quad t \geq 0.$$

The phase field model for pitting corrosion in [29] is composed by the following two equations,

$$\frac{\partial \phi}{\partial t}(\mathbf{x},t) = -L\left(\frac{\partial f}{\partial \phi} - \alpha_\phi \Delta \phi\right), \tag{1}$$

$$\frac{\partial c}{\partial t}(\mathbf{x},t) = \nabla \cdot M \nabla \left(\frac{\partial f}{\partial c} + \alpha_c \Delta c\right), \tag{2}$$

known as the Allen-Cahn and Cahn-Hilliard equations, respectively. The parameters α_ϕ and α_c in (1)–(2) are gradient energy coefficients associated with ϕ and c, respectively (see [29] for more details). In practice, only one of the diffusive terms is

*The preface of this special issue to which the article belongs is given in [7].

[a]Department of Mathematics, University of Salerno, Via Giovanni Paolo II n. 132, 84084 Fisciano (SA), Italy

Parameter	Physical meaning
L	Interface kinetics coefficient
α_ϕ/α_c	Gradient energy coefficient associate to ϕ/c
M	Diffusion mobility
A	Free energy curvature
D_c	Diffusion coefficient
c_L	Normalised equilibrium coefficient for liquid state
ω	Height of double well potential

Table 1.1: Parameters in the phase field corrosion model [18, 29].

sufficient to approximate the energy contribution from the diffuse interface; thus for simplicity $\alpha_c = 0$ is often assumed. The parameters L and M are the interface kinetics coefficient and the diffusion mobility for mass transport. The latter can be evaluated as a constant value,

$$M = \frac{D_c}{2A}, \tag{3}$$

where D_c the Fick's diffusion coefficient and A is the free energy curvature, assumed to be the same for both the solid and liquid phases.

The governing equations of the generic phase field model, (1)–(2), are finally specified by choosing a suitable expression for the free energy $f(\phi, c) : [0,1]^2 \mapsto \mathbb{R}$. For simulating the pitting corrosion in dilute solution, and looking at each point as a mixture of two phases with different concentration but same potential, the following expression has been considered in [27, 29]

$$f(\phi, c) = A(c - h(\phi)(1-c_L) - c_L)^2 + \omega g(\phi), \qquad h, g : [0,1] \mapsto \mathbb{R} \tag{4}$$

where $c_L \in \mathbb{R}$ is a normalised equilibrium concentration for the electrolytic solution, h is an infinitely differentiable and monotonically increasing function interpolating $h(0) = 0$ and $h(1) = 1$, g is a double well potential function, and ω is a real number. As in [29], we set $h(\phi) = -2\phi^3 + 3\phi^2$ and $g(\phi) = \phi^2(1-\phi)^2$.

Substituting (3) and (4) in (1)–(2), and defining,

$$D_\phi = L\alpha_\phi, \qquad F_1(\phi, c) = -L\left(2A(c_L - 1)[c - h(\phi)(1-c_L) - c_L]h'(\phi) + \omega g'(\phi)\right), \qquad F_2(\phi) = -(1-c_L)h(\phi), \tag{5}$$

yields the system of parabolic PDEs,

$$\frac{\partial \phi}{\partial t}(\mathbf{x}, t) = D_\phi \Delta\phi + F_1(\phi, c), \tag{6}$$

$$\frac{\partial c}{\partial t}(\mathbf{x}, t) = D_c \Delta(c + F_2(\phi)), \tag{7}$$

associated with suitable initial data and boundary conditions of Dirichlet or Neumann type.

All parameters introduced in the model are collected in Table 1.1 together with their physical interpretation.

The implicit treatment of the interface dynamics is a clear advantage of phase field models, as it avoids to explicitly detect the moving interface configuration within a finite element or finite difference implementation. However, this clashes with the high computational cost for the numerical solution of these models on large spatial domains and long integration time. Efficient numerical schemes for solving the phase field model for pitting corrosion in [29] based on a finite difference and finite elements space discretization, respectively, coupled with an exponential method in time have been proposed in [18] and [19]. In this paper we introduce a MATLAB implementation of this method that further improves its efficiency and enormously reduces the computational costs for the solution of proposed test experiments. The algorithm relies on the use of a MATLAB function described in [32, 33] that uses Krylov subspaces of adapted dimension for the computation of the effect of an exponential matrix of vectors. The implementation suggested in [18] uses instead a function in the well-known `Expokit` package [35] that uses Krylov subspaces of fixed dimension.

With these premises, the paper is organised as follows. In Section 2 we describe the numerical scheme in [18] for the phase-field model (6)–(7). The MATLAB programs for the solution of the scheme and for the graphic visualization of the solution are presented in Section 3. The proposed implementation is used for the solution of two benchmark problems in Section 4 and its efficiency is tested. Final remarks are presented in Section 5.

2 Numerical scheme

When the parameters in equations (6)–(7) are calibrated in order to accurately simulate realistic situations, the values of the diffusion coefficients D_ϕ and D_c are typically several order of magnitude smaller than the reaction term F_1 in (6). This implies that equation (6) is extremely stiff and conventional time integrators, such as the backward differentiation formula, require extremely small time steps to converge.

For this reason, we consider here the second-order accurate numerical method in [18], based on a finite difference space approximation coupled with an exponential Rosenbrock-Euler method in time. The use of the exponential method allows for the solution of the system with relatively large time steps. However, it requires the calculation of large sparse matrix exponentials that can be computationally very demanding. Nevertheless, since only equation (6) is stiff, then the exponential method can be used only for the solution of this equation, whereas a cheaper classic midpoint method can be used for the time integration of equation (7). Further computational savings are achieved by decoupling the two equation.

2.1 Space discretization

Let us consider a rectangular domain in a two dimensional space, $\Omega = [0, L_x] \times [0, L_y]$ and a uniform mesh defined by the nodes

$$\mathbf{x}_{i,j} := (x_i, y_j) = (i\Delta x, j\Delta y), \qquad i = 0, \dots, m_x + 1, \quad j = 0, \dots, m_y + 1.$$

We establish the following order relation between the nodes:

$$(x_{i_1}, y_{j_1}) < (x_{i_2}, y_{j_2}) \text{ if } j_1 < j_2, \text{ or if } j_1 = j_2 \text{ and } i_1 < i_2. \tag{8}$$

Let be $U_{i,j}$ the approximation of a function u at the node $\mathbf{x}_{i,j}$. We consider the classical centred difference approximation of the Laplacian,

$$\Delta U(\mathbf{x}_{i,j}) \approx \frac{U_{i-1,j} - 2U_{i,j} + U_{i+1,j}}{\Delta x^2} + \frac{U_{i,j-1} - 2U_{i,j} + U_{i,j+1}}{\Delta y^2}.$$

The matrix representation of the discrete Laplace operator above is given by

$$\mathcal{M} = I_{m_y} \otimes \mathcal{M}_x + \mathcal{M}_y \otimes I_{m_x},$$

where I_d is the identity matrix of order d, and, for $z \in \{x, y\}$,

$$\mathcal{M}_z = \frac{1}{\Delta z^2} \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \in \mathbb{R}^{m_z \times m_z},$$

if Dirichlet boundary conditions are assigned, or

$$\mathcal{M}_z = \frac{1}{\Delta z^2} \begin{bmatrix} -2 & 2 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 2 & -2 \end{bmatrix} \in \mathbb{R}^{m_z \times m_z},$$

if Neumann boundary conditions are assigned.

Let $\Phi(t)$ and $C(t)$ denote the vectors in $\mathbb{R}^{m_x m_y}$ of the approximations of $\phi(\mathbf{x}, t)$ and of $c(\mathbf{x}, t)$, respectively, at the nodes. These are ordered as entries of these vectors according to the ordering in (8). In other words, $\Phi_i(t) \approx \phi(x_{r+1}, y_{q+1}, t)$, and $C_i(t) \approx c(x_{r+1}, y_{q+1}, t)$, where q and r are the quotient and the residual in the division $(i-1)/m_x$, respectively. If the boundary conditions are homogeneous the semidiscretization of model (6)–(7) is given by

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = D_\phi \mathcal{M}\Phi + F_1(\Phi, C), \tag{9}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = D_c \mathcal{M}(C + F_2(\Phi)). \tag{10}$$

Otherwise, for $z \in \{\phi, c, F_2\}$, we denote with $\lambda_z = \lambda_z(t)$ the vector including the contributions given by the boundary values of function z in the diffusion terms. These are sparse vectors whose i-th entry is zero if $i = m_x q + r + 1$ with $0 < r < m_x - 1$ and $0 < q < m_y - 1$, since in this case the node (x_{r+1}, y_{q+1}) is an internal grid point. Otherwise, the i-th entry of $\lambda_z(t)$ depends on the value of function z at the boundary node (x_{r+1}, y_{q+1}) at time t. The semidiscrete model is in this case,

$$\frac{\mathrm{d}\Phi}{\mathrm{d}t} = D_\phi(\mathcal{M}\Phi + \lambda_\phi) + F_1(\Phi, C), \tag{11}$$

$$\frac{\mathrm{d}C}{\mathrm{d}t} = D_c\left(\mathcal{M}(C + F_2(\Phi)) + \lambda_c + \lambda_{F_2}\right). \tag{12}$$

In equations (9)-(10) and (11)-(12), functions F_1 and F_2 are defined according to (5) and act entry-wise on vector arguments.

Depending on the metal morphology the finite difference semidiscretization needs to be modified in order to apply the boundary conditions at suitable interface nodes.

2.2 Time discretization

We first define the exponential Rosenbrock-Euler method considering a generic autonomous system of ODEs,

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = F(u(t)), \qquad u(0) = u^0, \qquad u(t) \in \mathbb{R}^N,$$

where F is a regular function. Exponential integrators are based on a splitting of function F such that

$$\frac{\mathrm{d}u(t)}{\mathrm{d}t} = J_n(u(t)) + G(u(t)), \qquad t \in [t_n, t_{n+1}] \tag{13}$$

where $t_{n+1} = t_n + \Delta t$, $J_n = \frac{\partial F(u)}{\partial u}|_{u=u(t_n)}$ is the Jacobian matrix, and $G = F - J_n$.

Applying the variation-of-constant formula to (13), yields

$$u(t_{n+1}) = \mathrm{e}^{J_n \Delta t} u(t_n) + \int_0^{\Delta t} \mathrm{e}^{J_n(\Delta t - \tau)} G(u(t_n + \tau))\, \mathrm{d}\tau.$$

The exponential Rosenbrock-Euler method is then obtained by approximating $G(u(t_n + \tau)) = G(u(t_n))$ for $\tau \in [0, \Delta t]$, and gives the approximation

$$u^{n+1} = \mathrm{e}^{J_n \Delta t} u^n + \Delta t \varphi_1(J_n \Delta t) G(u^n), \tag{14}$$

where $\varphi_1(z) = z^{-1}(\mathrm{e}^z - 1)$. Method (14) can be equivalently written in the form

$$u^{n+1} = u^n + \Delta t \varphi_1(J_n \Delta t) F(u^n). \tag{15}$$

The numerical method in [18] relies on a variable splitting approach where formula (15) is applied only to the stiff equation (9), whereas a low-cost implicit scheme is used to approximate equation (10).

Let be ϕ^n and c^n the vectors of the approximations to $\phi(\mathbf{x}, t_n)$ and to $c(\mathbf{x}, t_n)$, respectively, with $t_n = n\Delta t$ and Δt a fixed stepsize.

By combining Rosenbrock-Euler method to solve equation (9) with the implicit midpoint method to solve equation (10), the fully discrete scheme in [18] amounts to

$$\begin{aligned}
\phi^{n+1} &= \phi^n + \Delta t \varphi_1(J_n \Delta t)(D_\phi \mathcal{M} \phi^n + F_1(\phi^n, \hat{c}^{n+1/2})), \qquad J_n = D_\phi \mathcal{M} + \frac{\partial F_1}{\partial \phi}(\phi^n, \hat{c}^{n+1/2}), \qquad n \geq 1 \\
c^{n+1} &= c^n + \Delta t D_c \mathcal{M}\left(\frac{c^{n+1} + c^n}{2} + F_2\left(\frac{\phi^{n+1} + \phi^n}{2}\right)\right),
\end{aligned}$$

where the value $\hat{c}^{n+1/2}$ is given by the extrapolation

$$\hat{c}^{n+1/2} = \frac{3}{2} c^n + \frac{1}{2} c^{n-1}. \tag{16}$$

As these formulae define a two-step method, an initialization procedure is needed. Approximations at time t_1 are obtained from

$$\begin{aligned}
\phi^1 &= \phi^0 + \Delta t \varphi_1(J_0 \Delta t)(D_\phi \mathcal{M} \phi^0 + F_1(\phi^0, c^0)), \qquad J_0 = D_\phi \mathcal{M} + \frac{\partial F_1}{\partial \phi}(\phi^0, c^0), \\
c^1 &= c^0 + \Delta t D_c \mathcal{M}\left(c^1 + F_2\left(\phi^1\right)\right),
\end{aligned}$$

where ϕ^0 and c^0 are given by the initial conditions. The scheme so defined is second-order accurate.

When the boundary conditions are non-homogeneous but independent of time, the time discretization of equations (11)–(12) is given by

$$\phi^{n+1} = \phi^n + \Delta t \varphi_1(J_n \Delta t)(D_\phi(\mathcal{M}\phi^n + \lambda_\phi) + F_1(\phi^n, \hat{c}^{n+1/2})), \qquad J_n = D_\phi \mathcal{M} + \frac{\partial F_1}{\partial \phi}(\phi^n, \hat{c}^{n+1/2}), \qquad n \geq 1 \tag{17}$$

$$c^{n+1} = c^n + \Delta t D_c \left(\mathcal{M}\left(\frac{c^{n+1} + c^n}{2} + F_2\left(\frac{\phi^{n+1} + \phi^n}{2}\right)\right) + \lambda_c + \lambda_{F_2}\right), \tag{18}$$

$$\phi^1 = \phi^0 + \Delta t \varphi_1(J_0 \Delta t)(D_\phi(\mathcal{M}\phi^0 + \lambda_\phi) + F_1(\phi^0, c^0)), \qquad J_0 = D_\phi \mathcal{M} + \frac{\partial F_1}{\partial \phi}(\phi^0, c^0), \tag{19}$$

$$c^1 = c^0 + \Delta t D_c \left(\mathcal{M}\left(c^1 + F_2\left(\phi^1\right)\right) + \lambda_c + \lambda_{F_2}\right). \tag{20}$$

The decoupling of the governing equations is a key point for the efficient solution of system (17)–(20). In fact, it allows solving first equation (17) (or (19)) to calculate ϕ^{n+1}. Subsequently, this value can be substituted in (18) (resp. (20)) that reduce to a linear system yielding the values of c^{n+1}.

3 MATLAB implementation

The definition of a MATLAB efficient algorithm relies on efficient techniques to evaluate function φ_1 in (17) and (19) and to solve the linear systems given by (18) and (20).

In this section we describe the MATLAB implementation of scheme (17)–(20), reporting the codes used to evaluate, analyse and visualize the approximate solutions.

3.1 Main program

We first describe the main program, `pitcorr.m`, focusing on input and output parameters and on the auxiliary functions for its implementation.

Input parameters

- `Lx`, `Ly` - double scalars
  Endpoints defining the domain of integration $\Omega = [0, Lx] \times [0, Ly]$.
- `T` - double scalar
  Endpoint of the time integration interval $t \in [0, T]$.
- `mx`, `my`, `N` - double scalars
  Number of internal nodes along x, y and t dimension, respectively.
- `f` - function handle
  Function that defines the diffusion coefficients D_ϕ and D_c and the functions F_1 and F_2 in (6) and (7). It computes also the boundary values of F_2 and the Jacobian matrix of F_1.
- `phiapp` - string
  String with name of the function that computes $\Delta t \varphi_1(J_n \Delta t)$ in equation (17) (and similarly in equation (19) for $n = 0$). It must be either `phiapp = 'phiv'` or `phiapp = 'expmvp'`.
- `initbound` - function handle
  Function that returns the initial and boundary values and the diffusion matrices $\mathcal{M}$ in equations (17)–(20).

Output parameters

- `phi,c` - double matrices
  Matrices of dimension $(mx \cdot my) \times (N+1)$ with the approximations of ϕ and c, respectively, at every $\mathbf{x} \in \Omega$ and $t \in [0, T]$.
- `X,Y` - double matrices
  Matrices of dimension $my \times mx$ defining the spatial grid.
- `t` - double array
  Vector of the time nodes having dimension $N + 1$.

The two matrices `phi` and `c` of the numerical solution are such that $\phi_{i,j} \approx \phi(x_{r+1}, y_{q+1}, t_{j-1})$ and $c_{i,j} \approx c(x_{r+1}, y_{q+1}, t_{j-1})$, where q and r are the quotient and the residual, respectively, in the division $(i-1)/m_x$. In other words, the j-th column is the vector with the approximations at time t_{j-1} ordered according to (8).

Auxiliary functions

In [18] the package `Expokit` [35] is used to compute the action of a matrix exponential on a vector. In this paper we consider two different functions to evaluate the effect of function φ_1 on a vector:

- `phiv.m` in the package `Expokit` [35];
- `expmvp.m` and `expmnorm.m` in [32, 33].

Function `phiv` takes as input a double scalar t, a double matrix J, and two double vectors v and u, and returns

$$z = \exp(tJ)v + t\varphi_1(tJ)u. \tag{21}$$

Function `expmvp`, instead, takes as input a double scalar t, and two double matrices J and W with columns W_j, $j = 1, \ldots, N$, and it returns

$$z = \exp(tJ)W_1 + t\varphi_1(tJ)W_2 + \Delta t^2 \varphi_2(tJ)W_3 + \ldots + t^{N-1}\varphi_{N-1}(tJ)W_N, \tag{22}$$

where the functions φ_n, $n > 1$ are defined as in [32, 33].

Both these functions do not compute the matrix exponential in isolation but directly its action on the operand vectors, and the calculations involve matrix-vector products only (matrix-free methods). The effect of function φ_1 on a vector is equivalently calculated by solving linear ODEs by means of adaptive time-stepping technique combined with a Krylov subspace approach.

The crucial difference between the two algorithms is that in `phiv.m` the size of the Krylov subspace is fixed, by default to thirty, whereas in `expmvp.m` the dimension of the Krylov subspace is adapted following the approach in [23]. Function `expmvp.m` is to be used in combination with `expmnorm.m`, a function that computes the matrix exponential using the scaling and squaring algorithm with a Padé approximation in [22]. It is equivalent to the MATLAB inbuilt function `expm.m`, except that the norm of the matrix to exponentiate is also an output, and this prevents to recompute the norm inside `expmvp.m`. Both `expmvp.m` and `expmnorm.m` can be downloaded from Niesen webpage*.

For efficiently solving the linear systems given by (18) and (20) we use the MATLAB inbuilt functions:

- `bicgstab.m`;
- `bicgstabl.m`.

Both these two functions implement a biconjugate gradients stabilized method. Function `bicgstab` has been used to solve the pitting corrosion problem already in [18]. Function `bicgstabl` has been introduced more recently to prevent stagnating situations where `bicgstab` does not converge. Both functions take as input a double matrix `A` a double array `b` and a double scalar `tol`. The double output command `[x,flag]=bicgstab(A,b,tol)`, returns the solution of `Ax=b` with tolerance `tol`, and a flag that specifies whether the algorithm successfully converged. The single output command `x=bicgstab(A,b,tol)` instead displays diagnostic messages that may slow down the iterations.

The main program, `pitcorr.m`, takes as input two further functions that must be defined by the user:

*`http://www1.maths.leeds.ac.uk/~jitse/software.html`

- `initbound.m`
  Input parameters
  - `Lx` and `Ly` - double scalars
    Endpoints defining the domain of integration $\Omega = [0, Lx] \times [0, Ly]$.
  - `mx` and `my` - double scalars
    Number of internal nodes along x and y dimension, respectively.
  - `dx` and `dy` - double scalars
    Step size along x and y dimension, respectively.
  - `x` and `y` - double arrays
    Vectors of the internal nodes along x and y dimension, respectively.

  Output parameters
  - `BoundVal` - double matrix;
    Matrix `BoundVal = [Boundxphi Boundyphi Boundxc Boundyc]` with dimension $(\texttt{mx} \cdot \texttt{my}) \times 4$. The entries of the first column, `Boundxphi`, are zero if the corresponding node (x, y) is not adjacent to the bottom or top side of the boundary, i.e. if $y \notin \{\Delta y, L_y - \Delta y\}$. Otherwise, the value is given by the boundary value of ϕ at the adjacent point on the boundary, i.e., $\phi(x, 0, t)$ or $\phi(x, L, t)$, respectively. The second column, `Boundyphi`, is defined in a similar way by considering the left and right side of the boundary instead, so interchanging the roles of x and y. The last two columns, `Boundxc` and `Boundyc`, are defined analogously for the boundary values of c.
  - `Mphi` and `Mc` - double matrices;
    Diffusion matrices `Mphi` and `Mc` that may not coincide if ϕ and c do not satisfy boundary conditions of the same type (Dirichlet/Neumann) on the entire boundary. However, usually this is not the case and `Mphi` and `Mc` typically coincide.
  - `phi0` and `c0` - double arrays;
    Vectors with the initial values of ϕ and c, respectively, at the internal nodes.
- `f.m`
  Input parameters
  - `phi` and `c` - double arrays;
    Vectors with last computed approximations of ϕ and c, respectively.
  - `dx` and `dy` - double scalars;
    Step size along x and y dimension, respectively.
  - `BoundVal` (optional) - double matrix;
    Matrix with boundary values of ϕ and c obtained from `initbound.m`.

  Output parameters - Four-input option
  - `F1` - double array;
    Vector of the reaction function F_1 computed on `phi` and `c` given in input.
  - `dF1` - double matrix;
    Jacobian matrix $\partial F_1(\phi, c)/\partial \phi$ calculated on `phi` and `c`.
  - `Dcoeff` - double array.
    Vector of the two diffusion coefficients D_ϕ and D_c in equations (6)–(7).

  Output parameters - Five-input option
  - `F1` - double array;
    Vector of the diffusion function F_2 computed on `phi` given in input.
  - `dF1` - double array;
    Jacobian matrix, $\partial F_2(\phi)/\partial \phi$, calculated on `phi`.

  Note that the main program described so far can be used to solve any problem in the form (6)–(7) with generic definitions of functions F_1 and F_2. In fact, these must be returned by the user-defined function `f.m`. In Section 4 we solve a couple of test examples considering the definitions in (5).

Description of the main program

The main program of the algorithm, `pitcorr.m`, is illustrated in Figure 3.1.

The first part of the code (lines 5-7) computes the space-time grid and initializes the unknown matrices. Subsequently (line 11), function `initbound` sets the boundary conditions, the diffusion matrices, and the initial data. The boundary values are then used (lines 12-13) to define the vectors λ_ϕ and λ_c in (17)–(20).

The next blocks of instructions solve equations (19)-(20) and, in a for loop, (17) and (18) until reaching the final time, T.

First, function `f.m` is used to define function F_1, its Jacobian and the diffusion coefficients in (6)–(7).

Depending on the value of `phiapp` (see lines 20-24), function `phiv` or `expmvp` is used to compute the effects of function φ_1 on the operand vectors given in equations (19) and (17). The right hand side of (19) is computed either by using `phiv` (see equation (21)) with input the time step `dt`, the Jacobian matrix `J`, and the two vectors

$$v = 0, \qquad u = D_\phi(\mathcal{M}\phi^0 + \lambda_\phi) + F_1(\phi^0, c^0),$$

```
function [phi,c,X,Y,t] = pitcorr(Lx,Ly,T,mx,my,N,f,phiapp,initbound)

% Grid and initialization

x = linspace(0,Lx,mx)'; y = linspace(0,Ly,my)'; t = linspace(0,T,N); [X,Y] = meshgrid(x,y);
dx = Lx/(mx-1); dy = Ly/(my-1); dt = T/(N-1);
phi = zeros(mx*my,N); c = zeros(mx*my,N);

% Initial and boundary conditions

[BoundVal,Mphi,Mc,phi(:,1),c(:,1)] = initbound(Lx,Ly,mx,my,dx,dy,x,y);
lambdaphi = BoundVal(:,1)/dy^2+BoundVal(:,2)/dx^2;
lambdac = BoundVal(:,3)/dy^2+BoundVal(:,4)/dx^2;

% Solving equation (19)

[F1,dF1,Dcoeff] = f(phi(:,1),c(:,1),dx,dy);
u = Dcoeff(1)*(Mphi*phi(:,1)+lambdaphi)+F1;
J = Dcoeff(1)*Mphi + dF1;
if strcmp(phiapp,'expmvp')
   phi(:,2) = phi(:,1)+expmvp(dt,J,[sparse(zeros(size(u))),u]);
elseif strcmp(phiapp,'phiv')
   phi(:,2) = phi(:,1)+phiv(dt,J,u,sparse(zeros(size(u))));
end

% Solving equation (20)

Asys = speye(size(Mc))-dt*Dcoeff(2)*Mc;
[F2,lambdaF2] = f(phi(:,2),c(:,1),dx,dy,BoundVal);
bsys = c(:,1)+Dcoeff(2)*dt*(Mphi*F2+lambdaF2+lambdac);
[c(:,2),~] = bicgstab(Asys,bsys,1e-13);
Asys = speye(size(Mc))-dt*Dcoeff(2)/2*Mc;

for k = 3:N

    % Solving equation (17)

    chat = 3/2*c(:,k-1)-1/2*c(:,k-2);
    [F1,dF1] = f(phi(:,k-1),chat,dx,dy);
    J = Dcoeff(1)*Mphi + dF1;
    u = Dcoeff(1)*(Mphi*phi(:,k-1)+lambdaphi)+F1;
    if strcmp(phiapp,'expmvp')
        z = expmvp(dt, J, [sparse(zeros(size(u))), sparse(u)]);
    elseif strcmp(phiapp,'phiv')
        z = phiv(dt, J, u, sparse(zeros(size(u))));
    end
    phi(:,k) = phi(:,k-1)+z;

    % Solving equation (18)

    [F2,lambdaF2] = f((phi(:,k)+phi(:,k-1))/2,chat,dx,dy,BoundVal);
    bsys = c(:,k-1)+Dcoeff(2)*dt*(Mc*c(:,k-1)/2+Mphi*F2+lambdac+lambdaF2);
    [c(:,k),~] = bicgstab(Asys,bsys,1e-13);

end
```

Figure 3.1: Main program `pitcorr.m`

or, otherwise (see equation (22)), by `expmvp` evaluated on the stepsize, `dt`, the Jacobian matrix `J`, and the two-column matrix

$$[0, u], \qquad u = D_\phi(\mathcal{M}\phi^0 + \lambda_\phi) + F_1(\phi^0, c^0).$$

Lines 28-30 define the linear system in equation (20) that yield the values of c^1. To solve it we use (line 31) function `bicgstab` with tolerance 10^{-13}, using the double output command without storing the flag value.

Before entering the for loop, the coefficient matrix for solving the linear systems in (18) is defined once for all in line 32.

The extrapolated value $\hat{c}^{n+1/2}$ in equation (16) is calculated in line 38. The following lines calculate iteratively the approximations of ϕ and c at each time step similarly as described in lines 17-31, by only modifying the formulae in lines 39, 51 and 52 to be in agreement with the right hand side of equations (17) and (18).

3.2 Supplementary programs

We describe here some programs that can be useful for the graphical representation and investigation of the solutions obtained from the main program, `pitcorr.m`, or as an alternative when memory limits prevent to store the solutions at every time step. All the auxiliary MATLAB functions used in the main or supplementary programs are listed in Table 3.1.

`pitcorr2.m`

A critical issue when solving pitting corrosion problems on large spatial domain and long integration times, is that a large number of nodes needs to be introduced and matrices `phi` and `c` may exceed MATLAB maximum array size preference, causing MATLAB to be unresponsive. To deal with these situations, we propose here an alternative program, `pitcorr2.m` in Figure 3.2, that does not store the solution at every time step, but it returns only the solution at the initial and final time. In this case, the size of the output matrices `phi` and `c` is $(mx \cdot my) \times 2$. The program can be suitably modified to store the solution at an arbitrary number of time steps.

`pit_anim.m`

For visualizing graphically the solutions calculated by `pitcorr.m`, we propose here the program `pit_anim.m` in Figure 3.3. This function takes as input arguments, respectively, the space grid `X`, `Y`, the vector of the time nodes `t`, a solution matrix `phi` and an optional scalar parameter `rec`.

The solution matrix `phi` in input can be either a two-column matrix with the approximations of ϕ and c at the final (or a specific) time, respectively, or the full solution `phi` (or `c`) at every time step returned by `pitcorr.m`.

In the first case, function `pit_anim.m` creates two figures with the 2D graphs of ϕ and c at the final (or prescribed) time. The columns of the matrix `phi` are reshaped into matrices having the same size of the computational grid (lines 5-6). The graphs are created by using the command `surf` (lines 7 and 9). Colours may be personalized by changing the argument of `colormap`. The option `shading interp` applies a piecewise bilinear shading, where the colour in each patch varies linearly and interpolates the corner values. The commands `view(2)` and `axis equal` in lines 8 and 10, set the default 2D view and the aspect ratio of the axes, respectively.

If the matrix `phi` includes instead the approximations of ϕ (or c) at each time step, an animation showing the evolution of the solution at every k-th step is displayed. The value of k is given by the integer variable '`step`' (line 14). The animation is created by displaying a sequence of graphs of the solutions at different stages, generated in a for loop (lines 21-33). The time interval between two consecutive frames is the computation time plus 0.1 sec (line 29). The graph of the solution at the final time is shown at the end also when k does not divide the number of time iterations (line 21).

If the optional input parameter `rec` $= 1$, then this animation is recorded in a video, "myvideo.avi", saved in the MATLAB current folder.

`pitdepth.m`

Program `pitdepth.m` in Figure 3.4 tracks the advance of the lowest descending pitting front above a certain height, $H_{\min}$. It is assumed that the pit either has a circular shape or it is the whole top edge (see examples in Section 4). This function takes as input the space grid, `X, Y`, the time nodes, `t`, the solution matrix, `phi` (or `c`), returned by `pitcorr.m`, and a double array, `pit`, with three entries: the height of the pit at the initial time, its radius, and the value $H_{\min}$ (line 4 in Figure 3.4). In the case of a circular pit, the initial height is intended as that of its center, and in the case of the corrosion of the whole edge, the radius must be set equal to zero. As an output it returns a double array, `ypit`, having as entries the location of the advancing pit at every time step.

The entries of `ypit` are computed by measuring the depth of the lowest point higher than $H_{\min}$ on the contour line corresponding to $\phi = 0.5$ (line 15). The contour line is calculated by using function `contour` (see line 8). To prevent the return of an empty set, the contour lines for $\phi = 0$ and $\phi = 1$ are also computed assuming that there is at least one point in the solution that is totally in the liquid or solid phase, as is usual. The output `h` is a handle to the contour object. In line 9 this is an input to function `getContourLineCoordinates` [15] that extracts in a table the (x, y) coordinates from contour lines organised by contour level and group. The table is converted into a matrix, `M`, and the ordinates of the points of the contour line for $\phi = 0.5$ are isolated in the array `A` (see lines 10 and 11).

At the very first time steps, if the spatial grid is not fine enough, it is possible that the corrosion front has not reached any grid point yet or that there is no node where the corrosion level $\phi = 0.5$ has been reached yet. In these cases, we approximate the location of the advancing front as its initial value (line 13).

At the end, in lines 18–19, a plot showing `ypit` against $\sqrt{t}$ is produced.

```
function [phi,c,X,Y,t] = pitcorr2(Lx,Ly,T,mx,my,N,f,phiapp,initbound)
% Grid and initialization

x = linspace(0,Lx,mx)'; y = linspace(0,Ly,my)'; t = linspace(0,T,N); [X,Y] = meshgrid(x,y);
dx = Lx/(mx-1); dy = Ly/(my-1); dt = T/(N-1);
phi = zeros(mx*my,2); c = zeros(mx*my,2);

% Initial and boundary conditions

[BoundVal,Mphi,Mc,phi(:,1),c(:,1)] = initbound(Lx,Ly,mx,my,dx,dy,x,y);
lambdaphi = BoundVal(:,1)/dy^2+BoundVal(:,2)/dx^2;
lambdac = BoundVal(:,3)/dy^2+BoundVal(:,4)/dx^2;

% Solving equation (19)

[F1,dF1,Dcoeff] = f(phi(:,1),c(:,1),dx,dy);
u = Dcoeff(1)*(Mphi*phi(:,1)+lambdaphi)+F1;
J = Dcoeff(1)*Mphi + dF1;
if strcmp(phiapp,'expmvp')
   phi(:,2) = phi(:,1)+expmvp(dt,J,[sparse(zeros(size(u))),u]);
elseif strcmp(phiapp,'phiv')
   phi(:,2) = phi(:,1)+phiv(dt,J,u,sparse(zeros(size(u))));
end

% Solving equation (20)

Asys = speye(size(Mc))-dt*Dcoeff(2)*Mc;
[F2,lambdaF2] = f(phi(:,2),c(:,1),dx,dy,BoundVal);
bsys = (c(:,1)+Dcoeff(2)*dt*(Mphi*F2+lambdaF2+lambdac));
clast = c(:,1);
[c(:,2),~] = bicgstab(Asys,bsys,1e-13);
Asys = speye(size(Mc))-dt*Dcoeff(2)/2*Mc;

for k = 3:N

    % Solving equation (17)

    chat = 3/2*c(:,2)-1/2*clast;
    [F1,dF1] = f(phi(:,2),chat,dx,dy);
    J = Dcoeff(1)*Mphi + dF1;
    u = Dcoeff(1)*(Mphi*phi(:,2)+lambdaphi)+F1;
    if strcmp(phiapp,'expmvp')
        z = expmvp(dt, J, [sparse(zeros(size(u))), sparse(u)]);
    elseif strcmp(phiapp,'phiv')
        z = phiv(dt, J, u, sparse(zeros(size(u))));
    end
    philast = phi(:,2);
    phi(:,2) = phi(:,2)+z;


    % Solving equation (18)

    [F2,lambdaF2] = f((phi(:,2)+philast)/2,chat,dx,dy,BoundVal);
    bsys = c(:,2)+Dcoeff(2)*dt*(Mc*c(:,2)/2+Mphi*F2+lambdac+lambdaF2);
    clast = c(:,2);
    [c(:,2),~] = bicgstab(Asys,bsys,1e-13);

end
```

Figure 3.2: Main program `pitcorr2.m`

```
function pit_anim(X,Y,t,phi,rec)

mx = size(X,2); my = size(X,1);
if size(phi,2)==2     % Graphs of the solution at final time
    p = reshape(phi(:,1),[mx,my])';
    q = reshape(phi(:,2),[mx,my])';
    figure, surf(X,Y,p), colormap(jet), shading interp
    view(2), axis equal, xlim([X(1),X(end)]), ylim([Y(1),Y(end)])
    figure, surf(X,Y,q), colormap(jet), shading interp
    view(2), axis equal, xlim([X(1),X(end)]), ylim([Y(1),Y(end)])
 else   % Animation
    if nargin == 4, rec = 0;    end
    N = length(t);
    step = round((N-1)/30); % Frame frequency.
        if rec == 1
            set(gcf, 'Color','white')
            vidObj = VideoWriter('myvideo.avi');
            vidObj.Quality = 100; vidObj.FrameRate = 10;
            open(vidObj);
        end
        for j = 1:step:ceil((N-1)/step)*step+1   % If step does not divide N-1 the view
                                                 % of the solution at final time is forced
            i = min(j,N);
            q = reshape(phi(:,i),[mx,my])';
            surf(X,Y,q), colormap(jet), shading interp
            set(gca,'Fontsize',14), xlabel('x'), ylabel('y')
            view(2), axis equal, xlim([X(1),X(end)]), ylim([Y(1),Y(end)])
            title(sprintf('Solution at t = %.2f',t(i)));
            pause(0.1)
            if rec == 1,
                writeVideo(vidObj, getframe(gcf));
            end
        end
        if rec == 1,
                close(gcf), close(vidObj);
        end
end
```

Figure 3.3: Function `pit_anim.m`

4 Numerical experiments

In this section we test the codes discussed in Section 3 considering two different problems and using MATLAB R2020b on a Windows desktop computer with a eight-core Intel 3.8 GHz Processor and 15.7 GB Memory.

We consider the pit corrosion problem (6)–(7) with functions F_1 and F_2 given in equation (5). The values of the parameters in equations (6)–(7) are set as in [18, 29], and are summarised in Table 4.1.

With these parameters, $D_\phi \sim 6.01 \cdot 10^{-6}$, whereas $-2AL(c_L - 1) \sim 2.06 \cdot 10^8$ and $-L\omega \sim -4.16 \cdot 10^6$. The very large order of magnitude of the ratio between the reaction term and the diffusion term in (6) makes the equation very stiff, justifying the use of an exponential method for its solution.

This problem is here solved by using the main program `pitcorr.m` with input `f=fcorr` illustrated in Figure 4.1 to set the parameters and functions at the right hand side of equations (6)–(7) and their derivatives.

Pencil electrode test

The first problem is a variant of the pencil electrode experiment in [17, 18, 29]. The specimen is a 300 μm long stainless steel (304 SS) with diameter $d = 25\mu$m. The specimen is wrapped in an epoxy resin with only the top and bottom ends exposed to an electrolyte solution. This configuration is modelled by setting $c = \phi = 0$ at the top and bottom edge of the boundary, whereas homogeneous Neumann boundary conditions are assigned along the rest of the boundary. We assume that at the initial time the specimen has not yet been attacked by corrosion anywhere, so we assign constant initial conditions $c^0 = \phi^0 = 1$ at all internal nodes.

We define a uniform grid with resolution $\Delta x = \Delta y = 1\mu\text{m} = 10^{-6}$ on $\Omega = [0, 25 \cdot 10^{-6}] \times [0, 300 \cdot 10^{-6}]$ and a time step $\Delta t = 10^{-3}$ to integrate till the final time $T = 1$.

```
function ypit=pitdepth(X,Y,t,phi,pit)

mx = size(X,2); my = size(X,1); N = length(t);  ypit = zeros(1,N);
pitloc = pit(1);pitrad = pit(2);Hmin=pit(3);
ypit(1) = pitloc-pitrad;
for i = 2:N
    q = reshape(phi(:,i),[mx,my])';
    [~, h] = contour(X, Y, q, [0,0.5,1]);
    contourTable = getContourLineCoordinates(h);
    M = contourTable{:,:};
    A = (M(:,1) == 0.5).*M(:,end);
    if isempty(min(A(A > Hmin)))
        ypit(i) = ypit(1);
    else
        ypit(i) = min(A(A > Hmin));
    end
end
plot(sqrt(t(1:end)),pitloc-ypit(1:end))
set(gca,'Fontsize',14), xlabel('$\sqrt{t}$','Interpreter','latex'), ylabel('Pit depth')
```

Figure 3.4: Function `pitdepth.m`

Function	Calling program	Scope
`phiv` [35]	`pitcorr/pitcorr2`	Effect of φ_1 on vectors using Krylov subspaces of fixed dimension
`expmvp` [32,33]	`pitcorr/pitcorr2`	Effect of φ_1 on vectors using Krylov subspaces of adapted dimension
`bicgstab/bicgstabl`	`pitcorr/pitcorr2`	Biconjugate gradients stabilized method to solve linear systems
`VideoWriter`	`pit_anim`	Constructs a VideoWriter object to write video data to an AVI file
`writeVideo`	`pit_anim`	Write video data to file
`getframe`	`pit_anim`	Returns a movie frame that is a snapshot of current axes
`colormap`	`pit_anim`	Sets the current figure's colormap
`shading interp`	`pit_anim`	Piecewise bilinear shading that linearly interpolates corner values
`contour`	`pitdepth`	Returns contour matrix and a handle to a contour object
`getContourLineCoordinates` [15]	`pitdepth`	Extracts coordinates from contour lines created by contour plots

Table 3.1: Auxiliary MATLAB functions.

Parameter	Value
L	2
A	$5.35 \cdot 10^{7}$
c_L	$\sim 3.57 \cdot 10^{-2}$
ω	$\sim 2.08 \cdot 10^{6}$
α_ϕ	$\sim 3.01 \cdot 10^{-6}$
D_c	$8.50 \cdot 10^{-10}$

Table 4.1: Parameters values in the phase field corrosion model [18,29].

Implementation details

This problem is solved by running script `test_example_1` in Figure 4.2. Lines 5–6 define the space-time domain and the computational grid. In line 10 we define the string `phiapp` that sets the function used to evaluate function φ_1 on vectors. We consider the two options `phiapp = 'phiv'` and `phiapp = 'expmvp'`.

In line 14 the initial and boundary values are assigned to `pitcorr` by function `initbound_1` in Figure 4.3. The initial conditions are assigned in lines 11–12 of Figure 4.3. Boundary conditions of Dirichlet or homogeneous Neumann type are defined by a cell vector with eight entries. The first four entries define the boundary conditions for ϕ on the bottom, left, top and right side, respectively. The last four entries define the boundary conditions for c in a similar way. Each cell defines either a Dirichlet boundary condition specified by a function of x (or y) or a homogeneous Neumann boundary condition if empty. In Figure 4.3 (lines 16-17) the boundary conditions in the cell array `BC` are of homogeneous Dirichlet type on the bottom and top edge, and of homogeneous Neumann type on the vertical edges for both ϕ and c, to model the configuration of the problem described above. However, by modifying array `BC` or the initial conditions, `phi0` and `c0`, this function can be used to solve different problems on a rectangular domain. The rest of the code computes the diffusion matrices and the boundary values accordingly. The program can be easily adapted to deal with non-homogeneous Neumann boundary conditions.

```
function [F1,dF1,Dcoeff] = fcorr(phi,c,dx,dy,BoundVal)

% Parameter and function defining F_2

cL = 5.1/143; h = @(phi) (-2*phi.^3+3*phi.^2);

if nargin > 4

    % Diffused function F_2 and boundary terms

    F1 = -h(phi)*(1-cL);
    dF1 = -(h(BoundVal(:,1))*(1-cL)/dy^2+h(BoundVal(:,2))*(1-cL)/dx^2);
else

    % Parameters

    L = 2; A = 5.35e7;
    alphaphi = 10*5e-6/(2.94*4*sqrt(2));
    omega = 100/(16*alphaphi);
    Dc = 8.5e-10;
    Dcoeff = [L*alphaphi,Dc];

    % Functions defining F_1 and derivatives

    dh = @(phi) -6*phi.^2+6*phi; ddh = @(phi) -12*phi+6;
    dg = @(phi) 2*phi.*(1-phi).^2-2*phi.^2.*(1-phi);
    ddg = @(phi) 2*(1-phi).^2-4*phi.*(1-phi)-4*phi.*(1-phi)+2*phi.^2;

    % Reaction function F_1 and derivative

    F1 = -L*(2*A*(cL-1)*(c-h(phi)*(1-cL)-cL).*dh(phi)+omega*dg(phi));
    dF1 = -L*(2*A*(cL-1)^2*dh(phi).^2+...
        2*A*(cL-1)*(c-h(phi)*(1-cL)-cL).*ddh(phi)+omega*ddg(phi));
    mxmy=length(phi);
    dF1=spdiags(dF1,0,mxmy,mxmy);
end
```

Figure 4.1: Function `fcorr.m`

Numerical results

In test_example_1, functions `tic` and `toc` are used to measure the computation time of the algorithm. We compare the results for the two choices `phiapp = 'phiv'` and `phiapp = 'expmvp'`. Using function `phiv.m`, it takes 194.88 seconds to compute the solution till the final time $T = 1$. If function `expmvp.m` is used instead, the computation time is of 25.28 seconds, which amounts to a reduction of the 87.0%. The maximum difference between the entries of the solution matrices computed using `phiv.m` and `expmvp.m` is $4.52 \cdot 10^{-11}$ for ϕ and $6.51 \cdot 10^{-11}$ for c, showing that although the computational cost of the two functions is very different, the results obtained are essentially the same. Finally, we test the efficiency of function `bigcstab.m` compared to the standard MATLAB backslash, '\', function for the solution of the linear systems (line 31 and 53 in Figure 3.1). Using `phiapp = 'expmvp'` and '\' the computation time increases to 33.09 seconds. The maximum difference between the two computed functions is $2.92 \cdot 10^{-11}$. This shows that the use of `bigcstab.m` indeed speeds up the whole computation without compromising accuracy. Using `bigcstabl.m` in place of `bigcstab.m` gives similar results. All details are summarised in Table 4.2.

We now solve the same numerical test till a longer time, $T = 225$. Also in this case, function `expmvp.m` combined with `bigcstab.m` records the lowest computation time, 7878 seconds (see Table 4.2). In Figure 4.4 we show the solution ϕ in the interior of the domain at four different time steps: $t = 0, 38, 152$ and 225. These graphs are equivalent to snapshots of a video generated by `pit_anim.m` illustrated in Figure 3.3. The results are consistent with those obtained in [18, 29] and with the laboratory experiments in [17] where only the top side of the specimen is subject to corrosion.

Figure 4.5 shows the evolution of the pit depth calculated by function `pitdepth.m` in Figure 3.4 with last input argument `pit=[Ly,0,Ly/2]`. The value `Hmin=Ly/2` has been set to prevent `ypit` to take values in the lowest part of the specimen, characterised by the presence of an ascending pit, and to actually track the descending pit only. The graph shows that the pit depth increases linearly with the square root of time consistently with the results in [17, 18, 29].

```
1  tic
2
3  % Defining domain and grid
4
5  Lx = 25e-6; Ly = 300e-6; T = 1;
6  mx = 26; my = 301; N = 1001;
7
8  % Set phiapp = 'phiv' or phiapp = 'expmvp'
9
10 phiapp = 'expmvp';
11
12 % Computing solution
13
14 [phi,c,X,Y,t] = pitcorr(Lx,Ly,T,mx,my,N,@fcorr,phiapp,@initbound_1);
15 toc
16
17 u=[phi(:,end) c(:,end)];  % Solution at the final time
18
19 pit_anim(X,Y,t,u)  % Graphs of phi and c at the final time
```

Figure 4.2: Script `text_example_1.m`

Table 4.2: Computation times (sec.) for solving `test_example_1` with $\Delta x = \Delta y = 10^{-6}$, and $\Delta t = 10^{-3}$.

	`phiv.m`	`expmvp.m`		
Final time	`bigcstab.m`	`bigcstab.m`	`bigcstabl.m`	`'\'`
$T = 1$	194.88	25.28	26.57	33.09
$T = 225$	48090.02	7878.37	8961.51	10430.29

Circular pit-growth

The second benchmark problem is the propagation of an isolated circular pit, inspired by the study of the evolution of a semi-circular pit growth in [17,18,29]. We consider a rectangular specimen of size 400μm$\times 200\mu$m ($\Omega = [0, 400\cdot 10^{-6}]\times[0, 200\cdot 10^{-6}]$) that at the initial time presents a circular pit with diameter 4μm situated at its center point, as shown on the top left of Figure 4.8.

This setting is modelled by assigning to system (6)–(7) homogeneous Dirichlet boundary conditions on the pit surface and homogeneous Neumann boundary conditions on the external boundary. As in [18,29] we set the same values of the parameters as before, listed in Table 4.1.

We consider again a spatial grid defined by $\Delta x = \Delta y = 1\mu$m$= 10^{-6}$, and solve till time $T = 1$ with time step $\Delta t = 10^{-3}$.

Implementation details

The solution of this problem is computed by the script `text_example_2.m` in Figure 4.6, whose description is similar to that of `text_example_1.m`. In this case the initial and boundary conditions are assigned by `initbound_2` in Figure 4.7.

In Figure 4.7 lines 5–12 define homogeneous Neumann boundary conditions on the external sides of the rectangular domain, initialise the diffusion matrix `Mphi` by neglecting that the diffusion can not happen inside the pit at the center of the domain, and define the computational grid in space. The rest of the code has the goal of modifying the diffusion matrix taking into account of the zero Dirichlet boundary conditions on the pit boundary.

In line 16 the central location of the pit and its radius are established. In line 17 we fix the initial conditions by assigning value 1 to the grid points outside the pit region and 0 otherwise. The initial conditions (equal for both ϕ and c) are reorganised into column vectors in line 18.

In lines 22–33 we identify the grid nodes in the liquid phase by finding their Cartesian coordinates. We first identify the position of these nodes in the vector `x` by looking at the nodes x_i such that $|x_i - \texttt{xpit}| < \texttt{pitrad}$ (line 22). Then for each x_i in this interval we need to identify the ordinates of the nodes that are internal to the pit. This is done by defining the array `inpit` (line 25) having entries 1 if the initial condition is 0 (liquid phase) or 0 otherwise (solid phase). If `inpit` turns out to be the zero vector, then there are no nodes internal to the pit and an error message is displayed to suggest the use of a finer grid (see lines 27-28). Otherwise, the points in the pit are identified by a sequence of consecutive 1 entries in `inpit`, that can be determined by the location of the first and last elements (lines 30–31).

The for loop in lines 38–42 identifies the position of the nodes in the pit according to the ordering (8) and casts the corresponding indexes in the vector `ind`. The non diffusion inside the pit is finally forced in the diffusion matrices in lines 43–44.

```
function [BoundVal,Mphi,Mc,phi0,c0] = initbound_1(Lx,Ly,mx,my,dx,dy,x,y)

ex = ones(mx,1); ey = ones(my,1);
Mx = spdiags([ex -2*ex ex], -1:1, mx, mx)/dx^2;
My = spdiags([ey -2*ey ey], -1:1, my, my)/dy^2;
Boundxphi = zeros(mx*my,1); Boundyphi = zeros(mx*my,1);
Boundxc = zeros(mx*my,1); Boundyc = zeros(mx*my,1);

% Initial configuration

phi0 = ones(mx*my,1);
c0 = ones(mx*my,1);

% Boundary conditions

BC={ @(x) zeros(size(x)),[],@(x) zeros(size(x)),[],...
@(x) zeros(size(x)),[],@(x) zeros(size(x)),[]};
DirBC = ~cellfun('isempty',BC); % 1 if Dirichlet, 0 if homogeneous Neumann

% We assume that on each side phi and c satisfy boundary conditions
% of the same type.
if DirBC(1) == 0
    My(1,2) = 2/dy^2; % Homogeneous Neumann
elseif DirBC(1) == 1
    Boundxphi = Boundxphi+kron([1;zeros([my-1,1])], BC{1}(x));
    Boundxc = Boundxc+kron([1;zeros([my-1,1])], BC{5}(x));
end
if DirBC(2) == 0
    Mx(1,2) = 2/dx^2; % Homogeneous Neumann
elseif DirBC(2) == 1
    Boundyphi = Boundyphi+kron(BC{2}(y),[1;zeros([mx-1,1])]);
    Boundyc = Boundyc+kron(BC{6}(y),[1;zeros([mx-1,1])]);
end
if DirBC(3) == 0
    My(end,end-1) = 2/dy^2; % Homogeneous Neumann
elseif DirBC(3) == 1
    Boundxphi = Boundxphi+kron([zeros([my-1,1]);1], BC{3}(x));
    Boundxc = Boundxc+kron([zeros([my-1,1]);1], BC{7}(x));
end
if DirBC(4) == 0
    Mx(end,end-1) = 2/dx^2; % Homogeneous Neumann
elseif DirBC(4) == 1
    Boundyphi = Boundyphi+kron(BC{4}(y),[zeros([mx-1,1]),;1]);
    Boundyc = Boundyc+kron(BC{8}(y),[zeros([mx-1,1]);1]);
end
Iy = speye(my); Ix = speye(mx);

% Diffusion matrices

Mphi = kron(Iy,Mx)+kron(My,Ix);
Mc = Mphi;

% Vector with boundary terms

BoundVal = [Boundxphi Boundyphi Boundxc Boundyc];
```

Figure 4.3: Function `initbound_1.m`

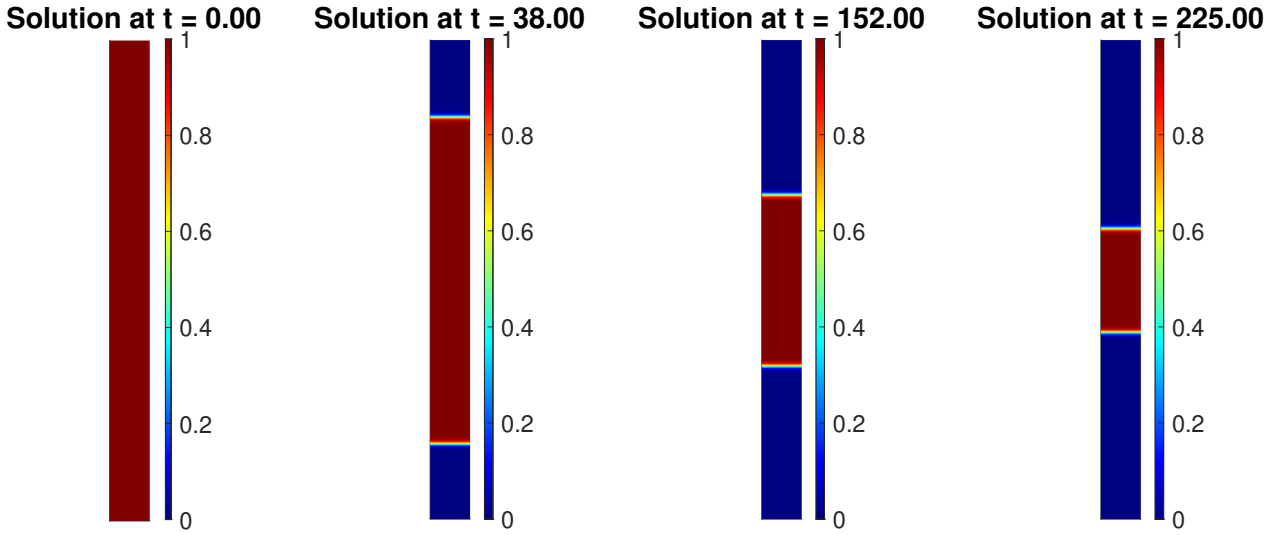

Figure 4.4: Configuration of ϕ given by `test_example_1` at different times.

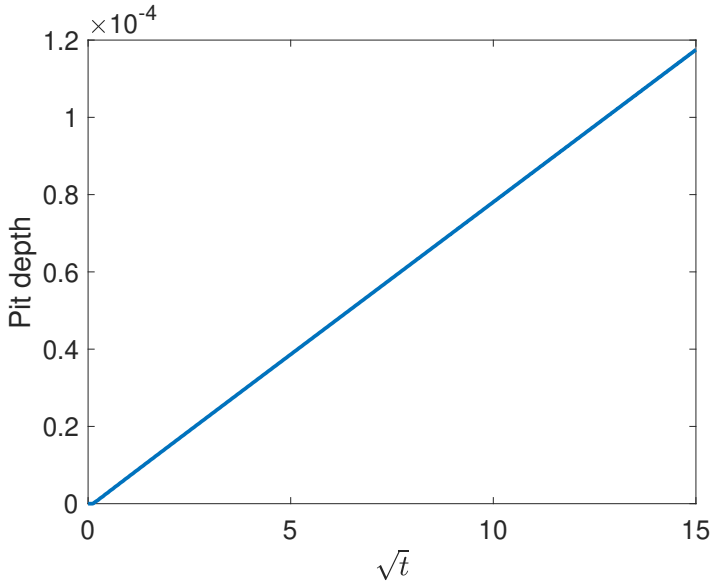

Figure 4.5: Evolution of the pit depth in the solution computed by `test_example_1`

Numerical results

Using function `phiv`, the solution is computed in 1545.33 seconds. If `expmvp` is used instead, the computation time reduces to 270.66 seconds, i.e. by about the 82.5%. The maximum difference between the two computed solutions in the whole time interval, is $4.78 \cdot 10^{-12}$ for ϕ and $6.41 \cdot 10^{-12}$ for c. Also in this case solving the linear system using function '\' instead of `bigcstab.m` increases the computation time to 394.64 seconds. The maximum difference between the solutions obtained is $2.21 \cdot 10^{-9}$, showing the higher efficiency of function `bigcstab.m`. As before, similar results are instead obtained by using `bigcstabl.m` in place of `bigcstab.m`. We summarise all computation times in Table 4.3.

As all previous tests have shown the high efficiency of `expmvp.m` combined with `bigcstab.m`, we only use this combination of functions to compute the solution till time $T = 100$. In this case, the matrices `phi` and `c` computed by `pitcorr` are too large (80601× 100001, requiring 60.1 GB of memory), and so we use instead `pitcorr2.m` in Figure 3.2 to obtain the solution at the final time. It takes 92407 seconds to solve this problem. The solution at times $t = 0, 20, 40, 100$ are presented in Figure 4.8 showing snapshots of a video recorded by using `pit_anim.m`.

Also in this case the pit depth growth is directly proportional to the square root of time as shown in Figure 4.9 and in agreement with the results in [17, 18, 29]. The graph has been realised by using function `pitdepth.m` in Figure 3.4 with `pit=[Ly/2,4e-6,0]` and it shows the pit depth evolution measured as the distance from the center of the pit for $t \in [0, 20]$.

```
tic

% Defining domain and grid

Lx = 400e-6; Ly = 200e-6; T = 1;
mx = 401; my = 201; N = 1001;

% Set phiapp = 'phiv' or phiapp = 'expmvp'

phiapp = 'expmvp';

% Computing solution

[phi,c,X,Y,t] = pitcorr(Lx,Ly,T,mx,my,N,@fcorr,phiapp,@initbound_2);
toc

u=[phi(:,end) c(:,end)];  % Solution at the final time

pit_anim(X,Y,t,u)  % Graphs of phi and c at the final time
```

Figure 4.6: Script `text_example_2.m`

Table 4.3: Computation times (sec.) for solving `test_example_2` with $T = 1$, $\Delta x = \Delta y = 10^{-6}$, and $\Delta t = 10^{-3}$.

`phiv.m`	`expmvp.m`		
`bigcstab.m`	`bigcstab.m`	`bigcstabl.m`	`'\'`
1545.33	270.66	274.52	394.64

5 Conclusions

In this paper we have developed an efficient MATLAB implementation for the numerical solution of a phase-field model for pitting corrosion of a metal specimen immersed in an electrolyte solution.

Models calibrated on the basis of laboratory experiments turn out to be very stiff. For this reason it is appropriate to consider exponential methods for their solution. We have discussed here an efficient implementation for these methods that relies on the use of the function `expmvp` in [32, 33]. This function computes the effect of matrix exponential functions on vectors by means of Krylov projection methods on subspaces of adapted dimension. We have shown that this makes the algorithm much more efficient than using the very popular function `phiv` in the `Expokit` package [35] based on Krylov subspaces of fixed dimension.

The MATLAB software has been illustrated and discussed in detail. The efficiency of the proposed procedure has been shown considering as test examples s pencil electrode experiment and the growth of an isolated circular pit.

Acknowledgements

D. Conte and G. Frasca-Caccia are members of the INdAM Research group GNCS. This work is supported by GNCS-INDAM project and by PRIN2017-MIUR project.

References

[1] K. V. Akpanyung, R. T. Loto. Pitting corrosion evaluation: a review. *J. Phys.: Conf. Ser.*, 1378:022088, 2019.

[2] D. Aregba-Driollet, F. Diele, R. Natalini. A mathematical model for the sulphur dioxide aggression to calcium carbonate stones: numerical approximation and asymptotic analysis. *SIAM J. Appl. Math.*, 64(5):1636–1667, 2004.

[3] C. Beckermann, H. J. Diepers, I. Steinbach, A. Karma, X. Tong. Modeling melt convection in phase-field simulations of solidification. *J. Comput. Phys.*, 154(2):468–496, 1999.

[4] M. A. Budroni, G. Pagano, D. Conte, B. Paternoster, R. D'Ambrosio, S. Ristori, A. Abou-Hassan, F. Rossi. Synchronization scenarios induced by delayed communication in arrays of diffusively coupled autonomous chemical oscillators. *Phys. Chem. Chem. Phys.*, 226:55–66, 2018.

[5] K. Burrage, A. Cardone, R. D'Ambrosio, B. Paternoster. Numerical solution of time fractional diffusion systems. *Appl. Numer. Math.*, 116:82–94, 2017.

[6] G. Califano, D. Conte. Optimal Schwarz waveform relaxation for fractional diffusion-wave equations. *Appl. Numer. Math.*, 127:125–141, 2018.

[7] R. Cavoretto, A. De Rossi. Software for Approximation 2022 (SA2022). *Dolomites Res. Notes Approx.*, Special Issue SA2022, 15:i–ii, 2022.

[8] L. Chen. Phase-field models for microstructure evolution. *Annu. Rev. Mater. Res.*, 32:113–140, 2002.

[9] F. Clarelli, B. De Filippo, R. Natalini. Mathematical model of copper corrosion. *Appl. Math. Model.*, 38(19-20):4804-4816, 2014.

```
function [BoundVal,Mphi,Mc,phi0,c0] = initbound_2(Lx,Ly,mx,my,dx,dy,x,y)

% Defining diffusion matrix

BoundVal = zeros(mx*my,4);
ex = ones(mx,1);ey = ones(my,1);
Mx = spdiags([ex -2*ex ex], -1:1, mx, mx)/dx^2;
My = spdiags([ey -2*ey ey], -1:1, my, my)/dy^2;
My(1,2) = 2/dy^2; Mx(1,2) = 2/dx^2; My(end,end-1) = 2/dy^2; Mx(end,end-1) = 2/dx^2;
Iy = speye(my); Ix = speye(mx);
Mphi = kron(Iy,Mx)+kron(My,Ix);
[X,Y] = meshgrid(x,y);

% Defining initial configuration

xpit = Lx/2;ypit = Ly/2;pitrad = 4e-6;
init = double((X-xpit).^2+(Y-ypit).^2>(pitrad)^2)';
phi0 = reshape(init,[],1); c0 = phi0;

% Finding coordinates of points in the pit

xloc = find(sparse((xpit-pitrad<x).*(x<xpit+pitrad)) == 1);
ylocmin = zeros(size(xloc)); ylocmax = zeros(size(xloc));
for i = 1:length(xloc)
    inpit = init(xloc(1)+i-1,:) == 0;
    if sum(inpit) == 0
       error(['There are no nodes in the initial configuration ' ...
            'of the pit. Use a finer space grid'])
    else
        ylocmin(i) = find(inpit,1);
        ylocmax(i) = find(inpit,1,'last');
    end
end

% Finding matrix indices of the nodes in the pit

cont = 1;
for i = 1:length(xloc)
    indsize = length(mx*(ylocmin(i)-1)+xloc(i):mx:mx*(ylocmax(i)-1)+xloc(i));
    ind(cont:cont+indsize-1) = mx*(ylocmin(i)-1)+xloc(i):mx:mx*(ylocmax(i)-1)+xloc(i);
    cont = cont+indsize;
end
Mphi(ind,:) = 0;
Mphi(:,ind) = 0;
Mc = Mphi;
```

Figure 4.7: Function `initbound_2.m`

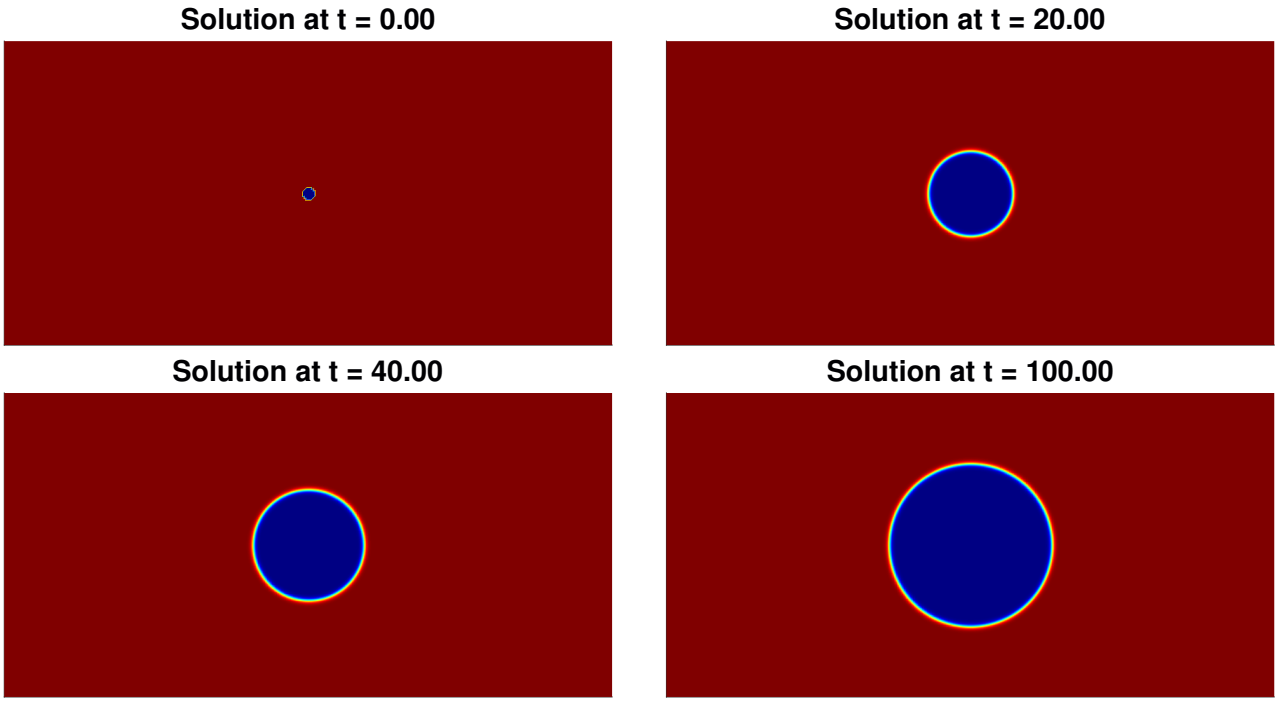

Figure 4.8: Configuration of ϕ given by `test_example_2` and `pitcorr_2` at different times.

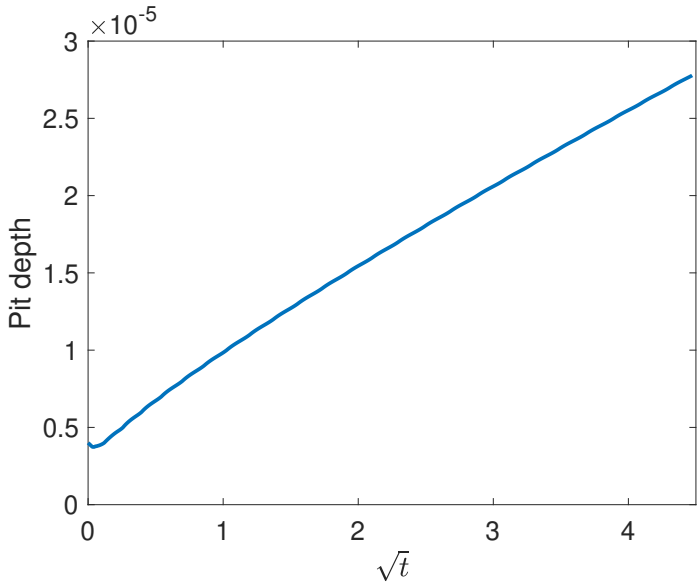

Figure 4.9: Evolution of the pit depth in the solution computed by `test_example_1` with $t \in [0, 20]$.

[10] B. Cockburn, C.-W. Shu. The local discontinuous Galerkin for time-dependent convection diffusion systems. *SIAM J. Numer. Anal.*, 35(6):2440–2463, 1998.

[11] A. Coco, M. Semplice, S. Serra Capizzano. A level-set multigrid technique for nonlinear diffusion in the numerical simulation of marble degradation under chemical pollutants. *Appl. Math. Comput.*, 386:125503, 2020.

[12] R. D'Ambrosio, M. Moccaldi, B. Paternoster. Adapted numerical methods for advection-reaction-diffusion problems generating periodic wavefronts. *Comput. Math. with Appl.*, 74(5):1029–1042, 2017.

[13] R. D'Ambrosio, B. Paternoster. Numerical solution of a diffusion problem by exponentially fitted finite difference methods. *SpringerPlus*, 3(1):425, 2014.

[14] R. D'Ambrosio, B. Paternoster. Numerical solution of reaction-diffusion systems of λ-ω Type by trigonometrically fitted methods. *J. Comput. Appl. Math.* 294:436–445, 2016.

[15] A. Danz. getContourLineCoordinates (`mathworks.com/matlabcentral/fileexchange/74010-getcontourlinecoordinates`), MATLAB Central File Exchange, (2022). Retrieved April 27, 2022.

[16] M. Donatelli, M. Semplice, S. Serra-Capizzano. AMG preconditioning for nonlinear degenerate parabolic equations on nonuniform grids with application to monument degradation. *Appl. Numer. Math.*, 68:1–18, 2013.

[17] P. Ernst, R. Newman. Pit growth studies in stainless steel foils. I. Introduction and pit growth kinetics. *Corros. Sci.*, 44:927–941, 2002.

[18] H. Gao, L. Ju, R. Duddu, H. Li. An efficient second-order linear scheme for the phase field model of corrosive dissolution. *J. Comput. Appl. Math.*, 367:112472, 2020.

[19] H. Gao, L. Ju, X. Li, R. Duddu. A space-time adaptive finite element method with exponential time integrator for the phase field model of pitting corrosion. *J. Comput. Phys.*, 406:109191, 2020.

[20] S. Gonzalez-Pinto, D. Hernandez-Abreu, S. Perez-Rodriguez. W-methods to stabilize standard explicit Runge-Kutta methods in the time integration of advection-diffusion-reaction PDEs. *J. Comput. Appl. Math.*, 316:143–160, 2017.

[21] S. Gonzalez-Pinto, D. Hernandez-Abreu, S. Perez-Rodriguez. AMFR-W-methods for parabolic problems with mixed derivatives. Applications to the Heston model. *J. Comput. Appl. Math.*, 387:112518, 2021.

[22] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM J. Matrix Anal. Appl.*, 26(4):1179-1193, (2005).

[23] M. Hochbruck, C. Lubich, H. Selhofer. Exponential integrators for large systems of differential equations. *SIAM J. Sci. Comput.*, 19(5):1552–1574.

[24] J. Hyman, J. Morel, M. Shashkov, S. Steinberg. Mimetic finite difference methods for diffusion equations *Comput. Geosci.*, 6:333–352, 2002.

[25] A. Karma, W. Rappel. Phase-field method for computationally efficient modeling of solidification with arbitrary interface kinetics. *Phys. Rev. E*, 53:R3017–R3020, 1996.

[26] J.B. Keller, S.I. Rubinow. Recurrent precipitation and Liesegang rings. *J. Chem. Phys.*, 74(9):5000-5007, 1981

[27] S. Kim, W. Kim, T. Suzuki. Phase-field model for binary alloys. *Phys. Rev. E*, 60:7186–7197, 1999.

[28] D. D. Macdonald, G. R. Engelhardt. 2.39 - Predictive modeling of corrosion. in: B. Cottis, M. Graham, R. Lindsay, S. Lyon, T. Richardson, D. Scantlebury, H. Stott (Eds.), Shreir's Corrosion, fourth ed., vol. 2:1630–1679, Elsevier, 2010.

[29] W. Mai, S. Soghrati, R. G. Buchheit. A phase field model for simulating the pitting corrosion. *Corros. Sci.*, 110:157–166, 2021.

[30] W. Navidi, Z. Shayer. An application of stochastic modeling to pitting of spent nuclear fuel canisters. *Prog. Nucl. Energy*, 107:48–56, 2018.

[31] S. Nesic, M. Nordsveen, N. Maxwell, M. Vrhovac. Probabilistic modelling of CO_2 corrosion laboratory data using neural networks. *Corros. Sci.*, 43:1373–1392, 2001.

[32] J. Niesen, W. Wright. A Krylov subspace method for option pricing. Available at *SSRN*: `https://ssrn.com/abstract=1799124`, 2011.

[33] J. Niesen, W. Wright. Algorithm 919: A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators. *ACM Trans. Math. Softw.*, 38(3):1–19, 2012.

[34] D. Rodney, Y. L. Bouar, A. Finel. Phase field methods and dislocations. *Acta Mater.*, 51(1):17–30, 2003.

[35] R. Sidje. Expokit: A software package for computing matrix exponentials. *ACM Trans. Math. Softw.*, 24:130–156, 1998.

[36] H. Singh, S. Nakabayashi. A temporal semi-stochastic model for pitting corrosion. *J. Electroanal. Chem.*, 766:60–70, 2016.

[37] A. Turnbull, D. A. Horner, B. J. Connolly Challenges in modelling the evolution of stress corrosion cracks from pits. *Eng. Fract. Mech.*, 76:633–640, 2009.

[38] A. Valor, F. Caleyo, L. Alfonso, J. C. VelÃizquez, J. M. Hallen. Markov chain models for the stochastic modeling of pitting corrosion. *Math. Probl. Eng.*, 2013:108386, 2013.

[39] A. A. Wheeler, B. T. Murray, R. J. Schaefer. Computation of dendrites using aphase field model. *Physica D*, 66(1):243–262, 1993.

[40] Y. Zhou, C. Zhang, L. Brugnano. An implicit difference scheme with the KPS preconditioner for two-dimensional time-space fractional convection-diffusion equations. *Comput. Math. Appl.*, 80:31–42, 2020.

[41] Y. Zhou, C. Zhang, L. Brugnano. Preconditioned quasi-compact boundary value methods for space-fractional diffusion equations. *Numer. Algorithms*, 84:633–649, 2020.