POLITECNICO DI MILANO
School of Industrial Engineering and Information
Master's Degree in Mathematical Engineering

A Scalable Positivity Preserving Solution Scheme for Coupled Nonlinear Advection-Diffusion-Reaction Problems

Supervisor: Prof. Carlo De Falco
Co-Supervisors: Prof. Luca Formaggia,
Pasquale Africa

Temenuzhka Valentinova Avramova
853568

Academic Year 2017-2018

Ai miei genitori,

Contents

List of Figures

List of Algorithms

Abstract

Often in mathematical modeling, we can encounter nonlinear problems with constrained variables, like, for example, the non-negativity constraint for concentration equations. We focus on three main steps for solving nonlinear systems of coupled partial differential equations, that are time discretization, space discretization and nonlinear method. We choose an adaptive time discretization, which preserves the first order of convergence, and a space one with a mesh ordered by p4est. Eventually, we apply a new projected Newton-Krylov method mixed with projected gradient direction. This setting has been tested on a nonlinear problem of tumor growth equations.

Sommario

E' piuttosto frequente incontrare problemi non lineari e, talvolta, le loro variabili possono essere vincolate, come, per esempio, il vincolo di non-negatività nelle equazioni di concentrazioni. In questo lavoro abbiamo rivolto la nostra attenzione sui principali passi per la risoluzione di sistemi di equazioni accopiate di diffusione-trasporto-razione, ovvero : discretizzazione temporale, discrtizzazione spaziale e scelta del metodo non lineare. Abbiamo optato per una discretizzazione in tempo adattiva, che ad ogni passo temporale scegliesse la lunghezza massima che assicura ordine di convergenza 1. Per la discretizzazione in spazio, abbiamo usato una mesh con ordinamento dei nodi secondo p4est, ideando anche delle strategie di parallelizzazione. Infine come metodo numerico, abbiamo implementato un metodo recentemnete proposto di Newton-Krylov proiettato con direzione del gradiente. L'insieme di queste scelte è stata infine testata su un modello non lineare di crescita del tumore.

Chapter 1

Tumor growth's model

1 Propose of the thesis

The main aim of our work is to propose a strategy to solve coupled nonlinear partial differential equations with constrained variables. Our attention will be focus on the main passages that are required for this kind of numerical solving, that are:

- time discretization,
- space discretization,
- nonlinear method,

where each point will be addressed in the next chapters.
There are many examples of problems with the features that we nominated, for example Turing systems, generally nonlinear, with variables that represent chemical concentration of two species, therefor constrained to be non-negative.
The example problem we chose is taken from article "On interfaces between cell populations with different mobilities" [11], because we know from previous studies on it, that it can be complicated enough to be a suitable test problem for validity and performances of our strategy. Let see how it is structured.

2 Tumor growth

2.1 Introduction to the model

We introduce a system of coupled time-depending advection-diffusion-reaction equations meat to describe the propagation of avascular tumor taken from [11].

$$\begin{cases} \partial_t m - \mu \, \nabla(m \nabla p) = G(p)m, \\ \partial_t n - \nu \, \nabla(n \nabla p) = 0. \end{cases} \tag{2.1}$$

Variable m indicates local density of *dividing cells*, therefore those of the cancer, while n is local density of healthy cells, considered *non-dividing* . Therefore both are constrained to be nonnegative. Pressure is given by a constitutive relation, in this way :

$$p := K_\gamma (n+m)^\gamma, \tag{2.2}$$

with $K_\gamma = \frac{\gamma+1}{\gamma}$, where γ is a positive parameter that controls the stiffness of pressure law. Coefficients $\mu > 0$ and $\nu > 0$ stand for mobility of dividing and non-dividing cells respectively.

Second term of the left-hand-side model is a recall of the Dracy's law since it suggests the tendency of cells to move down the pressure gradient.

Right-hand-side of first equation indicates reasonably that growth of division cells m is proportional to its own density trough the quantity $G(p)$. All this model is based on the observation that proliferating cells, m, exert a pressure on their neighbours, n, and the result is a cell motion, that makes the first ones push forward the seconds. However, because of competition for space, when the pressure is above a threshold, that it is called *homeostatic pressure* P_M, then the system enters in a quiescent state. For all this reasons, it is required that $G(P_M) = 0$ and $G'(p) < 0$, because when pressure increases, so the competition for space, the devision rate has to decrease.

As announced in [11], two cases has to be distinguished. When the dividing cells are more viscous ($\nu > \mu$), so characterized by less mobility, than non-dividing ones, it is supposed to generate segregation, that is m and n

separated through a sharp interface. Second case is the opposite one, $\mu > \nu$, that corresponds to the situation in which "more viscous fluid" expands in a "less viscous" one. This scenario is expected to generate instabilities.

2.2 One-dimensional traveling waves

Suppose that we are looking for solution of the form $m(x,t) = m(x - \sigma t)$ and $n(x,t) = n(x - \sigma t)$, with σ traveling wave's velocity. Let start with an initial condition in which m and n lie on supports and are strictly separated, that is : $Supp(m) = (-\infty, 0]$ and $Supp(n) = [0, r]$, with $r = 1$.
Then (2.1) becomes:

$$\begin{cases} -\sigma m' - \mu(mp')' = G(p)m, \\ -\sigma n' - \nu(np')' = 0 \end{cases} \tag{2.3}$$

The reaction term is chosen in this way $G(p) = P_M - p$. If we suppose that p' vanishes at $\pm\infty$, then from the second equation of (2.3), it is obtain:

$$(\sigma + \nu p')n = \sigma n_\infty, \tag{2.4}$$

where $n \to n_\infty$ for $x \to \infty$. In general we will take the case $n_\infty = 0$, consequently, from (2.4), it comes:

$$p_n(\xi) = \frac{\sigma}{\nu}(r - \xi), \tag{2.5}$$

with $\xi = x - \sigma t$ and $x \in [0, r]$. Since m is equal to 0 in the support of n, using (2.2), the solution of second equation in (2.3) is :

$$n(\xi) = \Big(\frac{p(\xi)}{K_\gamma}\Big)^{\frac{1}{\gamma}}.$$

While to find the solution of first equation in (2.3), we need to put ourselves in the case of large γ (incompressibility limit), that can be seen as $\gamma = \frac{1}{\epsilon}$, with $0 < \epsilon << 1$. Adding the equations of (2.3), it is obtained

$$-\sigma(m + n)' - ((\mu m + \nu n)p')' = mG(p). \tag{2.6}$$

In particular, on the support of m, it becomes:

$$-\sigma p' - \mu(p'^2 + \gamma p p'') = \gamma p G(p).$$

We look for a solution of the following form:

$$p_m(\xi) = P_0(\xi) + \epsilon P_1(\xi), \tag{2.7}$$

with $p(-\infty) = P_M$.
Now, if we integrate (2.6), it comes out : $-\sigma(m+n)(x) - ((\mu m + \nu n)p')(x) = \int_x^1 mG(p)$. By the continuity of m, n and, consequently, of p, and (2.5), this expression is obtained:

$$p'(0^-) = \frac{\nu}{\mu} p'(0^+) = -\frac{\sigma}{\mu}.$$

This is used as boundary conditions for (2.7), then, if we neglect the part multiplied by ϵ, we obtain :

$$p_m(\xi) = P_M + (\frac{\sigma}{\nu} - P_M) \exp\Big(\frac{\xi}{\sqrt{\mu}}\Big), \quad \text{with} \quad \sigma = \frac{P_M \sqrt{(\mu)}\nu}{r\sqrt{\mu} + \nu}. \tag{2.8}$$

The expression of $p_m(\xi)$ and $p_n(\xi)$ can be used as initial guess, putting $\xi = x$, and then, a way to validate the chosen numerical method, is to verify if the wave's velocity, that we see in the simulations, is the theoretical one of (2.8). Theory of this section is taken from [11], but there is also a contribution from professor Pasquale Ciarletta, that helped us finding the expression of the solution.

2.3 Bi-dimensional propagation

In paper [11] it is investigated also the behaviour of two-dimensional spherical waves with zero Neumann boundary conditions and initial conditions:

$$m(x, y, t = 0) := a_m e^{-b_m(x^2+y^2)} \quad \text{and} \quad n(x, y, t = 0) := a_n e^{-b_n(x^2+y^2)},$$

with

$$a_m = 0.1, \quad a_n = 0.8, \quad b_m = 5 \times 10^{-1}, \quad b_n = 5 \times 10^{-7}.$$

Furthermore, reaction term is defined in the following way

$$G(p) := \frac{200}{\pi} \arctan(4(P_M - p))_+, \quad P_M = 30,$$

with a domain of $[0, L] \times [0, L]$, with $L = 45$.
We are going to test both one and bi-dimensional case, taking settings used by them as a first test, and then trying to stimulate the problem in different ways in order to test out strategy of solving.

Chapter 2

Discretization of the problem

1 Time discretization

Let consider a general differential equation in the following form:

$$u' = f(t, u), \quad \text{with} \quad u(t_0) = u_0. \tag{1.1}$$

We can image this as a different way of writing a time depending partial differential equation, like those of our test problem (2.1) in the previous chapter. Indeed, u' represents $\partial_t m$ or $\partial_t n$, while $f(t, u)$ contains all the rest of the equations.
One common way to discretize in time is to use Eulero's backward method, that consists in :

$$u_{n+1} = u_n + \Delta_n t \; f(t_{n+1}, u_{n+1}), \quad \text{with} \quad u_i := u(t_i) \quad \text{and} \quad \Delta_n t = t_{n+1} - t_n.$$

Actually this is what we want to use, but in general we know that its order of convergence is only 1, which means that the norm of the difference between the real and the approximated solution is bounded by Δt through a constant,

$$||u - u_{ex}|| < C\Delta t. \tag{1.2}$$

Therefore, we expect that, if we ask high accuracy, a small time step has to be adopted, but this can be really expensive when we try to solve a nonlinear

problem with a not coarse mesh.
This is why we opted for a time adaptivity strategy, that uses a local extrapolation method which at each time step, essentially, gives the maximum length of $\Delta_n t$ that guarantee the prearranged accuracy. We will see that the convergence order will be again 1, but with a smaller constant C, that permits bigger time step.
This can happen if we construct two approximations of $u(t_{n+1})$ at each time step and, therefore, their difference is an estimate of the local error for the less precise result and can be used as a step size control. Let see how.

1.1 Truncation error

First of all, we will show how the difference of the two approximations is a good estimate of the local error.
Reminding that $u'_n := f(t_n, u_n)$ and presuming that the solution of the previous step $\hat{u}_n$ is exact, let set our guess in this way:

$$u^0_{n+1} := \hat{u}_n + \Delta_n t \; \hat{u}'_n. \tag{1.3}$$

This one represents out first approximation of the solution in t_{n+1}, while the second one is the following numerical solution:

$$u_{n+1} := \hat{u}_n + \Delta_n t \; f(t_{n+1}, u_{n+1}).$$

Local truncation error is the difference between exact and numerical solution in the current step, that is $\tau_{n+1} := \hat{u}_{n+1} - u_{n+1}$.
Keeping in mind Taylor expansion of the exact solution in t_{n+1},

$$\hat{u}_{n+1} := \hat{u}(t_{n+1}) = \hat{u}(t_n + \Delta_n t) = \hat{u}_n + \Delta_n t \; \hat{u}'_n + C\Delta_n t^2,$$

let compute the difference between the two approximations:

$$\begin{aligned} u^0_{n+1} - u_{n+1} = &\quad \hat{u}_n + \Delta_n t \; \hat{u}'_n - u_{n+1} \\ = &\quad \hat{u}_{n+1} - C\Delta_n t^2 - u_{n+1} \\ = &\qquad \tau_{n+1} - C\Delta_n t^2. \end{aligned}$$

This shows that the difference between the two approximations is a good estimate of the truncation error.

1.2 Time adaptivity

In practice we can not presume that we have the exact solution of the previous step, then (1.3) becomes

$$u^0_{n+1} := u_n + \Delta_n t \, \frac{u_n - u_{n-1}}{\Delta_{n-1} t}.$$

This means that the difference between u^0_{n+1} and u_{n+1} is not anymore the same order as τ_{n+1}, but we will see soon what happens. Let first see the time adaptivity's procedure in Algorithm 1.
What we notice is that Δt is computed at each step, in line 16, using the information $\frac{tol}{err}$, that indicates how much norm of the difference between initial guess and numerical solution, called error, is lower than the tolerance. Smaller is the error, bigger is the time step's length that we can afford. Safety parameters, as $facmax, facmin$ and fac, are chosen not to allow Δt do increase o decrease too fast. For example for the value of fac we have different proposal in [9], like 0.8, 0.9, $\sqrt{0.25}$ and $\sqrt{0.38}$ and we will use last one.

1.3 Order of convergence

Depending on the value of tol that we choose, Algorithm 1 will do a certain number N of time steps. Even if the steps will be different, we can consider $\frac{1}{N}$ as an indicator of the average step's length and what happens is that tol is of order 2, that means $tol \sim (\frac{1}{N})^2$. But what about the order of the error, that is , the difference between exact and numerical solution?
We decided to show it with an example:

$$u' = 10\cos(t) - 3u,$$

with exact solution $u_{ex} = \sin(t) + 3\cos(t)$.
We implemented Algorithm 1 with different values of tol and then we plotted

Algorithm 1: Time Adaptivity

$t = t_0$, $\Delta t \in \mathbb{R}^+$, $n = 0$
while $t < T$ **do**
  if $t + \Delta t > T$ **then**
    $\Delta t = T - t$
  end if
  $t = t + \Delta t$
  if $n < 1$ **then**
    $u^0_{n+1} = u_0$
  else
    $u^0_{n+1} = u_n + \Delta t \ \frac{u_n - u_{n-1}}{\Delta_{old} t}$
  end if
  solve $u_{n+1} = u_n + \Delta t \ f(t_{n+1}, u_{n+1})$.
  $err = ||u^0_{n+1} - u_{n+1}||$
  if $err < tol$ **then**
    $\Delta_{old} t = \Delta t$
    $\Delta t = \Delta t \cdot \min(facmax, \max(facmin, fac \cdot \sqrt{(\frac{tol}{err})}))$
    $n = n + 1$
  else
    $t = t - \Delta t$
    $\Delta t = \frac{\Delta t}{2}$
  end if
end while

the error and the values of tol respect to N with a logarithmic scale . In Figure 1.1 it is shown how the tolerance tol is related to N with a second order, while the error with a order one. We found in Section 1.1 that they should be of the same order, but as we said before, this is because we don't have the exact solution at each previous step and the loss of one order is what we have to pay. What do we gain with this procedure if the order is the same of backward Eulero?
In the same example, we test both normal Eulero backward and time adaptivity method, in order to compare how many time steps were needed by both of them to achieve the same accuracy. Figure 1.2 shows that with the time adaptivity method less time steps are required, more precisely they are circa 18% less for each test. This is because, as we anticipated before, the value of C in (1.2) is smaller with this technique.

2 Space discretization

The mesh that we decided to use has the quadtrees' numeration for the nodes (see [2]). Before going in more details, we will discuss about some difficulties that we have encountered.
In view of the fact that we have to solve nonlinear systems, we used the C++ library LIS (Library of Iterative Solvers for linear systems), that, as the name suggests, solves liner problems with iterative methods. Since we needed to implement parallel solving to make the computing time reasonable, we created an interface for LIS, `lis_distributed_class`, in the library that we used for our implementations, called BIM, that manage a "decentralized" parallelization. This means that if we consider a linear system $Au = f$, each processor will have only its piece of A, u and f. In particular LIS requires matrix A to be divided in strictly separated ranges of rows. Also vectors u and f have to be divided among processors with the same ranges. The point is that the numeration that we have chosen for the mesh, does not divide matrix and vectors as LIS requires in a parallel environment.

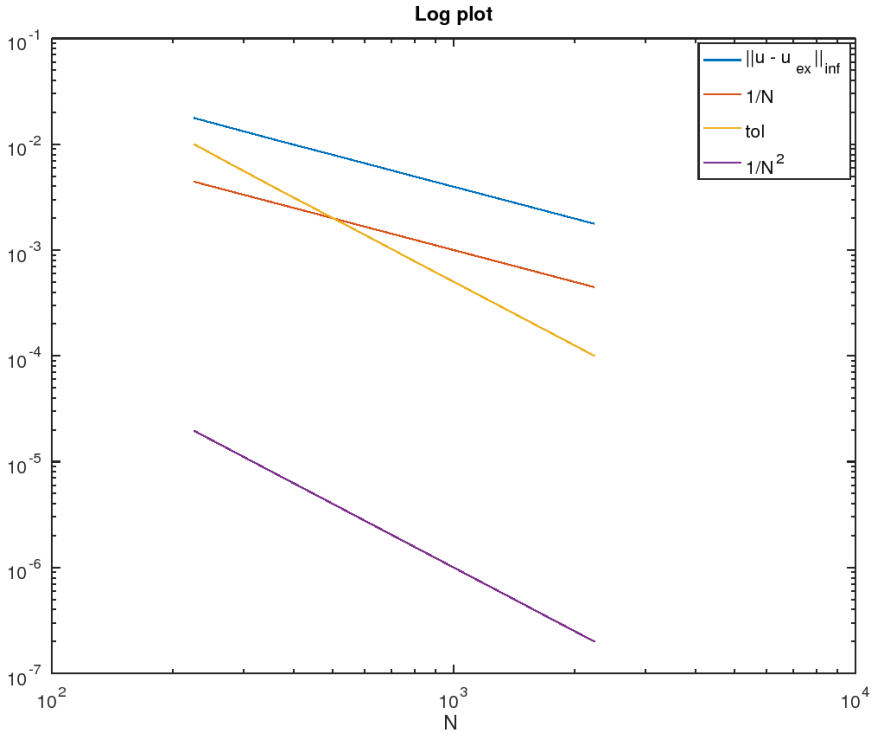


Figure 1.1: The logarithmic plot shows how *tol*, the tolerance on the difference between the two approximations of the solution, that are numerical solution and initial guess, is of second order respect to $\frac{1}{N}$. Instead, the error, that is the norm of the difference between exact and numerical solution, $||u - u_{ex}||_\infty$, is of order one.

Let see what happens and how we handled with it.

2.1 Numeration of quadtrees meshes

As we said above, solver LIS requires to have the matrix A divided by rows whenever it is asked parallel solving. For example, if we have a matrix with dimension 25×25 and we want to parallelize it with 2 processors, then the first processor will have the first 15 rows, while the second one the others 10.

Let see how the numeration with quadtrees works for an uniform mesh. Just to fix the ideas, imagine to have a mesh of 4×4 quadrants, that is 5×5

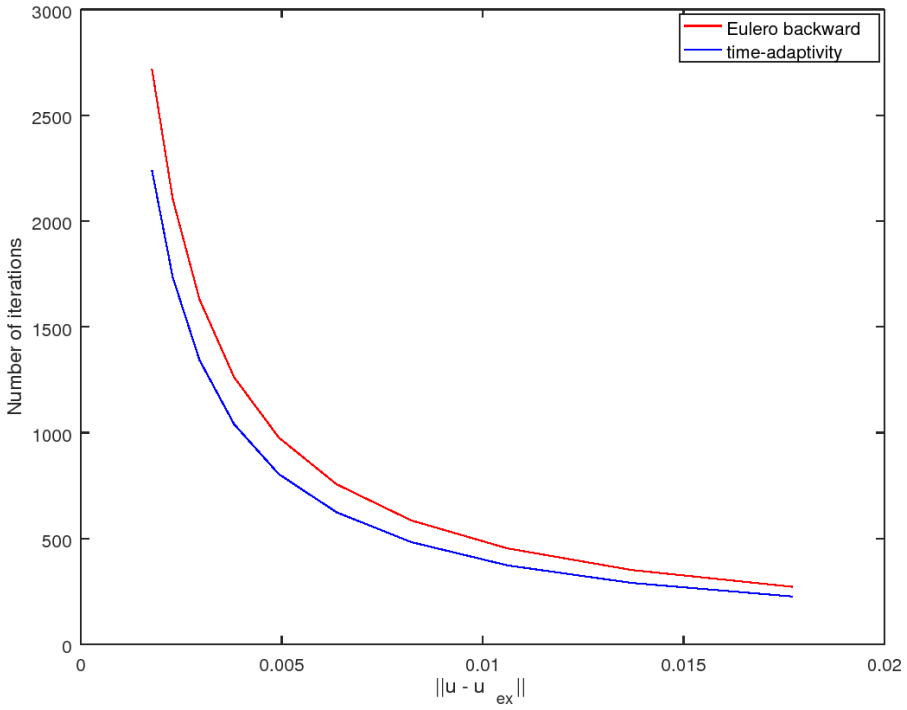
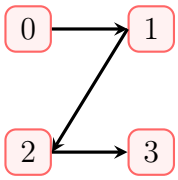


Figure 1.2: Order of convergence is one in both Eulero backward and time adaptivity methods, but, in the second case, the error $||u - u_{ex}||$ is proportional to $\frac{1}{N}$ with a smaller constant, that means less iterations can be done for achieve the same accuracy. In particular, in this case, they are circa 18%.

nodes. The numeration starts form the first quadrant of the first row from the top down and it is done in this order:



Nodes in all quadrants are numerated with this order and also the sequence of quadrants that are taken has the same logic, as is shown in Figure 2.1. With this procedure the final numeration of all the nodes of a 5×5 mesh results to be the one in Figure 2.2.

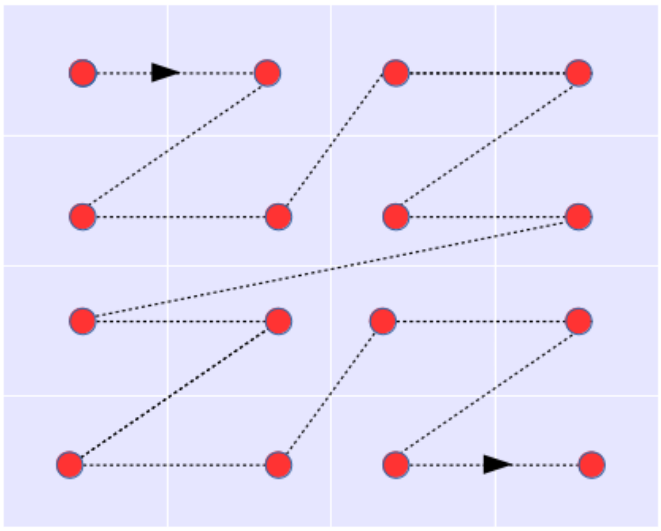


Figure 2.1: Sequence of quadrants taken to be numerated in a quadtrees mesh

With a mesh of this dimension, in discrete approximation all the space variables, will turn into vectors of $5^2 = 25$ components. Therefore the linear system that we have to solve has the shape $Au = f$, with $A \in \mathbb{R}^{25} \times \mathbb{R}^{25}$ and $u, f \in \mathbb{R}^{25}$. Since our system is coming from advection-diffusion-reaction equations, the most general pattern that matrix A can have is the one implemented by the method called `bim2a_structure (tmsh, A)` of class `sparse_matrix`. This pattern is supposed to compare each node with all its four neighbours, because the problem has diffusion term and, indeed A has to contain also the discrete gradient of the variable u. For example, in row 6, the previous method will allocate memory in A, in column 6, 2, 7 and 15. If we decide to solve the system with more processors, for example two, then the mesh in Figure 2.2 will be split in two parts. All the first 8 quadrants with their nodes will be of processor 1, and the other 8 will go to processor 2. Shared nodes, also called local nodes, that are nodes on the border of quadrants belonging to different processors, will be owned by the first processor that "touches" them. Therefore, in our example nodes 6, 7, 8, 13 and 14, will be owned by processor 1. But then, what happens to the matrix A ? Each

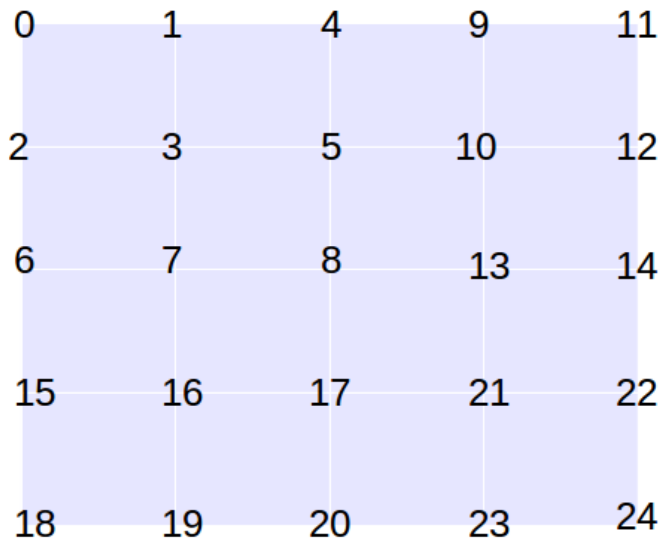


Figure 2.2: Quadtrees numeration of a 5×5 mesh. With two processors, the first 8 quadrants will go to processor 1, while the others 8 to processor 2. Nodes on the border (6,7,8,13,14) are called "shared" or "non local" and belong to the first processor that touches them, that is processor 1.

processor will have only the cells of the global matrix that link nodes owned by that processors or shared. In our case, processor 1 will have those cells of A that have indexes less or equal to the number of the last node owned by it, that is node 14. So, it will not have, for example, cells (14,22) or (6,15), that are actually present in rows 14 and 6 in the global matrix. Figure 2.3 shows how the patter of the global matrix is divided, in particular blu elements are of processor 1, while red of processor 2. As we can see there are overlaps that are all the red elements in the processor 1 's region. In fact, in those positions, both processors have their contribution, and to obtain the values of the global matrix, we have to sum them.

The point of this introduction of quadtrees mesh, actually, is to show that the natural separation among processors of the mesh's regions, so of A, is not divided by rows, and there are even overlaps for same elements. Having said that, we already know that LIS wants matrix A to be strictly separated by

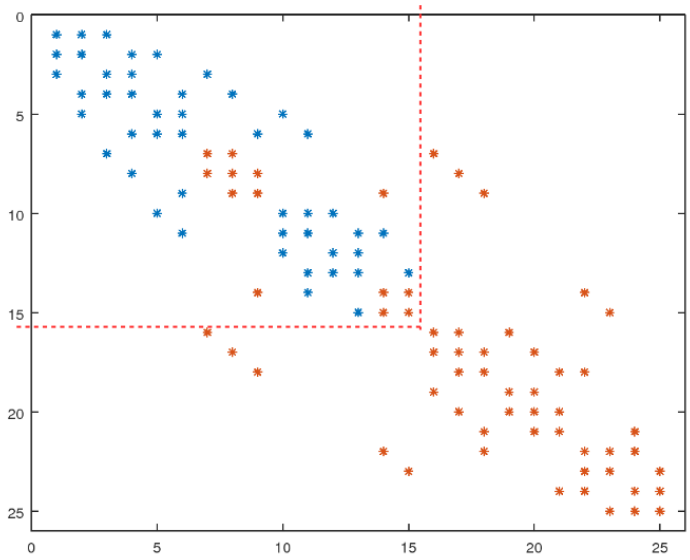


Figure 2.3: It is illustrated the pattern of global matrix A, blu cells are owned by processor 1 and red by processor 2. Where we see red elements in the region of processor 1, it means that there are overlaps, both processors has a value for that cells, and the sum of them gives the value of the global matrix. From this plot is clear that the division of a mesh with quadtrees numerations among processors is not made row by row, as LIS requires.

rows, so our next aim was to create a new version of the already existing class in BIM of sparse matrices called `sparse_matrix`, that handles this problem.

2.2 Parallelization on the mesh

Sparse matrix distributed class

In this section, we present the alternative of class `sparse_matrix`, that was created to allow each processor to assemble its matrix as LIS solver requires. This class is called `distributed_sparse_matrix` and its main variables are the following.

Listing 2.1: class distributed_sparse_matrix

```
1  class
2  distributed_sparse_matrix
3  : public sparse_matrix
4  {
5  private :
6    void
7    non_local_csr_index ();
8    void
9    non_local_csr_value ();
10   size_t is, ie;
11   MPI_Comm comm;
12   int mpirank, mpisize;
13   int nnz_owned;
14   struct
15   non_local_t
16   {
17     std::vector<int> prc_ptr, row_ind, col_ind;
18     std::vector<double> a;
19   } non_local;
20   std::map<int, std::vector<int>> row_buffers;
21   std::map<int, std::vector<int>> col_buffers;
22   std::map<int, std::vector<double>> val_buffers;
23   std::vector<int> ranges;
24   std::vector<int> rank_nnz;
25 public :
26   void
27   set_ranges (size_t is_, size_t ie_, MPI_Comm comm_ =
28               MPI_COMM_WORLD);
29   distributed_sparse_matrix (size_t is_, size_t ie_,
30                              MPI_Comm comm_= MPI_COMM_WORLD)
```

```
31      : mapped (false)
32      { set_ranges (is_, ie_, comm_); }
33    distributed_sparse_matrix (MPI_Comm comm_ = MPI_COMM_WORLD)
34      : comm (comm_), mapped (false)
35      { }
```

Let A be a matrix of type `distributed_sparse_matrix`. First of all, we have to set variables `is` and `ie`, which indicate the first and the last row owned by the current rank. These are computed in `lis_distributed_class` and passed to A by a method in the following way: `A.set_ranges (is, ie)`. This sets also the components of `ranges`, that is the rows' ranges in which the global matrix is divided among the processors.
In lines 15-19 it is defined the structure `non_local` that has `prc_ptr`, a vector supposed to contain the numbers of nonzero elements that local A has in each range, different from the one owned by the current processor. Vector `prc_ptr`, like all the others in `non_local`, changes from rank to rank; indeed each rank has a different A. So this is a way to indicate how many elements and to who the current rank has to communicate. The other vectors in `non_local` are basically the indexes and values in AIJ format of the elements of A out of the range of current rank. All of them are computed by `non_local_csr_index`.

Listing 2.2: distributed_sparse_matrix::non_local_csr_index ()

```
1  void
2  distributed_sparse_matrix::non_local_csr_index ()
3  {
4    non_local.row_ind.clear ();
5    non_local.col_ind.clear ();
6    non_local.a.clear ();
7    this->set_properties ();
8    non_local.prc_ptr.assign (this->mpisize + 1, 0);
9    sparse_matrix::col_iterator jj;
```

```
10    for (size_t ii = 0; ii < this->mpisize; ++ii)
11      {
12        non_local.prc_ptr[ii+1] = non_local.prc_ptr[ii];
13        if (ii != mpirank)
14          for (size_t kk = ranges[ii]; kk < ranges[ii+1]; ++kk)
15            {
16              non_local.prc_ptr[ii+1] += (*this)[kk].size ();
17              non_local.col_ind.reserve ((*this)[kk].size ());
18              non_local.row_ind.reserve ((*this)[kk].size ());
19              for (jj  = (*this)[kk].begin ();
20                jj != (*this)[kk].end (); ++jj)
21                {
22                  non_local.row_ind.push_back (kk);
23                  non_local.col_ind.push_back
24                                  (this->col_idx (jj));
25                  non_local.a.push_back (this->col_val (jj));
26                }
27            }
28      }
29    non_local.a.resize (non_local.row_ind.size ());
30  }
```

As we see, `prc_ptr` is, actually, the cumulative sum of the number of nonzero elements in the others ranges. We also see that with the check in line 13, only out-of-range elements are considered.

Up to now, for each A implemented by different ranks, we can identify which elements of the matrix are out of the current processor's range and to which range they below, therefore to which rank they have to be communicated. Going back to Figure 2.3 and taking the ranges $(0, 14)$ for rank 0 and $(15, 25)$ for rank 1, matrix A of rank 0 will have empty vectors in `non_local` structure, because its elements are all in the current range, while rank 1 will have `non_local` vectors of size 18, since this is the number of its elements that

stay in rank 0 's range. Now the aim is to communicated these components to rank 0.

Listing 2.3: distributed_sparse_matrix::remap ()

```
1  void
2  distributed_sparse_matrix::remap ()
3  {
4    non_local_csr_index ();
5    /// Distribute buffer sizes
6    rank_nnz.assign (mpisize, 0);
7    for (int ii = 0; ii < mpisize; ++ii)
8    rank_nnz[ii] = non_local.prc_ptr[ii+1]
9                   - non_local.prc_ptr[ii];
10   MPI_Alltoall (MPI_IN_PLACE, 1, MPI_INT, &(rank_nnz[0]), 1,
11   MPI_INT, comm);
12   /// Allocate buffers
13   for (int ii = 0; ii < mpisize; ++ii)
14     if ((rank_nnz[ii] > 0) && (ii != mpirank))
15       {
16         row_buffers[ii].resize (rank_nnz[ii]);
17         col_buffers[ii].resize (rank_nnz[ii]);
18         val_buffers[ii].resize (rank_nnz[ii]);
19       }
20   /// Communicate overlap regions
21   /// 1) communicate row_ptr
22   std::vector<MPI_Request> reqs;
23   for (int ii = 0; ii < mpisize; ++ii)
24     {
25       if (ii == mpirank) continue; // No communication to self!
26       if (rank_nnz[ii] > 0) // we must receive
27                             //something from rank ii
28         {
```

```
29            int recv_tag = ii    + mpisize * mpirank;
30            reqs.resize (reqs.size () + 1);
31            MPI_Irecv (&(row_buffers[ii][0]), row_buffers[ii].size (),
32                       MPI_INT, ii, recv_tag, comm, &(reqs.back ()));
33          }
34        int rank_nnz_snd_ii = non_local.prc_ptr[ii+1] -
35        non_local.prc_ptr[ii];
36        if (rank_nnz_snd_ii > 0) // we must send something to rank ii
37          {
38            int send_tag = mpirank + mpisize * ii;
39            reqs.resize (reqs.size () + 1);
40            MPI_Isend (&(non_local.row_ind[non_local.prc_ptr[ii]]),
41                       rank_nnz_snd_ii, MPI_INT, ii, send_tag, comm,
42                       &(reqs.back ()));
43          }
44      }
45    MPI_Waitall (reqs.size (), &(reqs[0]), MPI_STATUSES_IGNORE);
46                 reqs.clear ();
47    /// 2) communicate col_ind
48    for (int ii = 0; ii < mpisize; ++ii)
49      {
50        if (ii == mpirank) continue; // No communication to self!
51        if (rank_nnz[ii] > 0) // we must receive
52        // something from rank ii
53          {
54            int recv_tag = ii    + mpisize * mpirank;
55            reqs.resize (reqs.size () + 1);
56            MPI_Irecv (&(col_buffers[ii][0]), col_buffers[ii].size (),
57                       MPI_INT, ii, recv_tag, comm, &(reqs.back ()));
58          }
59        int rank_nnz_snd_ii = non_local.prc_ptr[ii+1] -
```

```
60          non_local.prc_ptr[ii];
61          if (rank_nnz_snd_ii > 0) // we must send
62          // something to rank ii
63            {
64              int send_tag = mpirank + mpisize * ii;
65              reqs.resize (reqs.size () + 1);
66              MPI_Isend (&(non_local.col_ind[non_local.prc_ptr[ii]]),
67                         rank_nnz_snd_ii, MPI_INT, ii, send_tag, comm,
68                         &(reqs.back ()));
69            }
70        }
71    MPI_Waitall (reqs.size (), &(reqs[0]), MPI_STATUSES_IGNORE);
72    reqs.clear ();
73    mapped = true;
74    update = true;
75  }
```

Here we encounter `row_buffers`, `col_buffers` and `val_buffers` that basically are containers of the elements that the current rank receives from the others. The numbers of these elements are set in the vector `rank_nnz` with `MPI_Alltoall` (lines 7-12). With this numbers, we set buffers' sizes .
Now comes the crucial part, in which each rank sends the indexes of its non local elements and receives the others' non local elements that are in its range. With line 25, communication to itself is avoided and the procedure of sending and receiving is done only if actually there is something to send or receive. Therefore, essentially, `distributed_sparse_matrix::remap ()` set the indexes of the cells in A in which each processor has to receive a value. For the communication of these values, we made the next method.

Listing 2.4: distributed_sparse_matrix::assemble ()

```
1  void
2  distributed_sparse_matrix::assemble ()
```

```
3  {
4    if (! mapped)
5      remap ();
6    non_local_csr_value ();
7    /// 3) communicate values
8    std::vector<MPI_Request> reqs;
9    for (int ii = 0; ii < mpisize; ++ii)
10     {
11       if (ii == mpirank) continue; // No communication to self!
12       if (rank_nnz[ii] > 0) // we must receive something
13       // from rank ii
14         {
15           int recv_tag = ii    + mpisize * mpirank;
16           reqs.resize (reqs.size () + 1);
17           MPI_Irecv (&(val_buffers[ii][0]), val_buffers[ii].size (),
18                      MPI_DOUBLE, ii, recv_tag, comm, &(reqs.back
19                      ()));
20         }
21     int rank_nnz_snd_ii = non_local.prc_ptr[ii+1] -
22     non_local.prc_ptr[ii];
23     if (rank_nnz_snd_ii > 0) // we must send something
24     // to rank ii
25       {
26         int send_tag = mpirank + mpisize * ii;
27         reqs.resize (reqs.size () + 1);
28         MPI_Isend (&(non_local.a[non_local.prc_ptr[ii]]),
29                    rank_nnz_snd_ii, MPI_DOUBLE, ii, send_tag,
30                    comm, &(reqs.back ()));
31       }
32     }
33   MPI_Waitall (reqs.size (), &(reqs[0]), MPI_STATUSES_IGNORE);
```

```
34                reqs.clear ();
35    /// 4) insert communicated values into sparse_matrix
36    for (int ii = 0; ii < mpisize; ++ii) // loop over ranks
37      if (ii != mpirank)
38        for (int kk = 0; kk < rank_nnz[ii]; ++kk)
39          (*this)[row_buffers[ii][kk]][col_buffers[ii][kk]]
40                                        += val_buffers[ii][kk];
41    /// 5) zero out communicated values
42    for (int iprc = 0; iprc < mpisize; ++iprc)
43      if (iprc != mpirank)
44        {
45          for (int ii = non_local.prc_ptr[iprc];
46                ii < non_local.prc_ptr[iprc+1]; ++ii)
47            // the following will throw if we try to access
48            // an element which does not exist yet!
49            (*this)[non_local.row_ind[ii]].at (non_local.col_ind[ii])
50                                                                = 0.0;
51        }
52    if (update)
53      {
54        nnz_owned = 0;
55        for (int irow = is; irow < ie; ++irow)
56          nnz_owned += (*this)[irow].size ();
57      }
58    update = false;
59  }
```

In nonlinear solving of partial differential equations problems, often, pattern of the matrix A remains the same for all the nonlinear and, eventually, time steps. This is why we actually separated the communications of the indexes from the one of the values; indeed, ones computed, `row_buffers` and `col_buffers` are supposed to remain the same, therefore there is no

need to calculate them every time. Actually, the same is true also for the `non_local` structure of A. This is why we implemented also the method `non_local_csr_value ()`, that updates only the values of `non_local.a`. Regarding the buffers, communication of the values is essentially the same as before and, it is in lines 36-40, that the communicated values are inserted in matrix A. At the end all the out-of-range values of A are set to zero.

Distributed vector class

In our example of linear system $Au = f$, we know that every rank has his own matrix A and that the sum of all of them represents the global matrix. Since $(A_1 + A_2)(u_1 + u_2) \neq A_1u_1 + A_2u_2$, each rank has to have the global values of u, but it is enough to allocate the ones that correspond to owned and shared nodes, and not to all the nodes, since A has nonzero elements only for these nodes. On the other hand, f could have elements different from zero only where A has nonzero rows, that is in all owned and shared nodes. But, again, if we want to pass this vectors in LIS for parallel solving, we have to pass just the piece of the current range, and it means that all the out-of-range values has to bo communicated to their correspondent processors. For this regard, we implemented a new class called `distributed_vector`. Let see how is structured.

Listing 2.5: distributed_vector class

```
1  class
2  distributed_vector
3  {
4  private:
5    MPI_Comm comm;
6    int mpirank;
7    int mpisize;
8    bool mapped;
9    int owned_count;
```

```
10    int is, ie;
11    /// vector entries owned by the current node
12    std::vector<double> owned_data;
13    /// vector entries touched by the current node
14    /// that belong somwhere else
15    std::map<int, double> non_local_data;
16    /// Structure to hold data of ghost entries
17    /// in a format amenable for send/receive
18    struct
19    ghosts_t
20      {
21        std::vector<int> prc_ptr, row_ind, rank_nnz;
22        std::vector<double> a;
23      } ghosts;
24    /// Copy data from non_local_data to ghosts
25    void
26    ghost_csr ();
27    /// Update ghosts.
28    void
29    ghost_csr_update ();
30    /// Structure to hold data of ghost entries
31    /// in a format amenable for send/receive
32    struct
33    mirrors_t
34      {
35        std::vector<int> prc_ptr, row_ind, rank_nnz;
36        std::vector<double> a;
37      } mirrors;
38    /// the entries that are owned by rank i
39    /// are numbered between ranges[i]
40    /// and ranges[i+1]
```

```
41    std::vector<int> ranges;
42    /// check whether the idx-th global
43    /// entry is owned by the current rank
44    inline bool
45    is_owned (int idx) const
46      { return idx >= is && idx < ie; }
47    /// check whether the idx-th global
48    /// entry is owned by the given rank
49    inline bool
50    is_owned (int idx, int irank) const
51      { return idx >= ranges[irank] && idx < ranges[irank+1]; }
52    /// return the rank that owns the idx-th entry
53    inline int
54    owner (int idx)
55      {
56        auto ir = std::lower_bound (ranges.begin (),
57                                    ranges.end (), idx);
58        return int ((ir - ranges.begin ()) - 1);
59      }
60  public:
61    distributed_vector (int owned_count_,
62    MPI_Comm comm_ = MPI_COMM_WORLD);
63    distributed_vector (int is_, int ie_,
64    MPI_Comm comm_ = MPI_COMM_WORLD);
65    distributed_vector () {};
66    void
67    set_owned_count (int owned_count_, MPI_Comm comm_ =
68                     MPI_COMM_WORLD);
```

First of all, let have a look at the constructors. Suppose to have a distributed vector u, when we declare it, we have also to provide either the dimension or the extremes of the owned nodes' range. If it is not possible, because, for

example we are declaring this kind of vectors in a class that does not have this information internally, then we can declared it without anything, but when we want to use that vector, we must implement first `u.set_owned_count (dimension_range)`. This is because the first two constructors and that method set the size of `owned_data`, which with `non_local_data`, is the main container of the class. Indeed, the first one is meant to contain values of the owned nodes, so the values that ere in the range of the current rank. The second will contain eventually the values and their indexes of out-of-range elements. Structures `ghosts` and `mirrors` has the same role as `non_local` and `buffers` in `distributed_sparse_matrix`. The first one, that contains the out-of-range elements, is computed with private method `ghosts_csr ()`, while the second one is computed after the communications done in methods called `remap ()` and `assemble ()`, analogue at the ones of the matrix's class, and contains the elements that are in the current range and are sent from the other ranks. All the mechanism of sending and receiving is very similar to the one shown in the previous section, so we will limit ourselves to recall the main steps.

1. copy `non_local_data` into ghosts structure;
2. send ghosts indexes and receive into mirrors;
3. send ghosts data and receive into mirrors;
4. add mirrors into `owned_data`;
5. copy `owned_data` into mirrors;
6. send mirrors data and receive into ghosts;
7. copy ghosts data into `non_local_data`.

The two first steps are of the method `remap ()`, while the others are in `assemble ()`. Besides, steps 5, 6 and 7 were not present in the previous class. They are a kind of inverse communication, indeed after the update

of `owned_data`, this one is communicated to the others' `non_local_data`. In this way, we ensure that also `non_local_data` will be updated with the global values. This needs to happen because, as we said before, each rank has to have the global values also out-of-the range of u.
The use of objects of this class is quite easy since is sufficient to write `u[i] = val` and, if `i` belong to the current range, then it is equivalent to write `owned_data[i] = val`, otherwise it is inserted a new component `<i, val>` in the map `non_local_data`, or it is updated, if one exists already with this index. It is enough to initialize all the out-of-range elements, that we are interested to keep inside u, at the beginning and then they will be always updated with `assemble ()`. In our case, they will be the shared nodes.

2.3 Ordering of more fields on the same mesh

Now we have to specify how the presence of two fields, as it is in our test problem (2.1) in Chapter 1, can be handled with the parallelization.
In our computational environment we keep the fields, m and n, in the same container, that in this case will have twice the mesh size. The same is true for all the other vectors, like the initial guess or the right-hand-side of the system.
Consequently we can imagine a partition as the one in Figure 2.4, with N number of nodes. This structure can be useful when we, for example, implement the matrix; indeed, generally, the four blocks A_{mm}, A_{mn}, A_{nm} and A_{nn} are done separately. But since our parallelization is based on the mesh's quadrants, we decided to weave the two fields m and n, which means that vectors will be ordered by nodes. Starting from the first node, all the variables evaluated in it are listed before passing to the second node and so on. Figure 2.5 shows the idea of this ordering, of course all the structure of A will be different from the one in Figure 2.4, but we can still treated A as separated in 4 blocks, even if it is not, using variables of type `ordering`. Indeed each field will have its `ordering` variable, for example `ord0` for m and

`ord1` for n. In practice these variables will contain the indexes in u of m and n, which are `ord0 = {0,2,4,...}` and `ord1 = {1,3,5,...}`. Therefore, for example, computing `A(ord0, ord0)`, means to pick block A_{mm}.

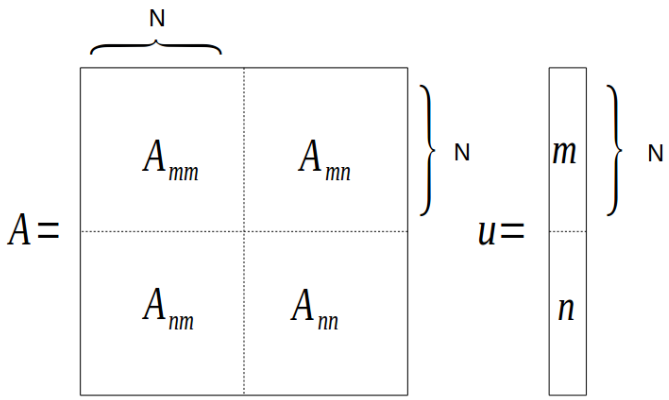


Figure 2.4: Partition of matrix and vectors among m and n, with N the number of mesh's nodes. This is an intuitive division, but is not going to be the chosen one because we want to order by nodes and not by fields. Nevertheless this is an useful way to imagine it; indeed when we compute the matrix, we will do it with each block separately. For working with the matrix as it is structure in this way, even if it is not, we implemented variables of type `ordering` that do this job.

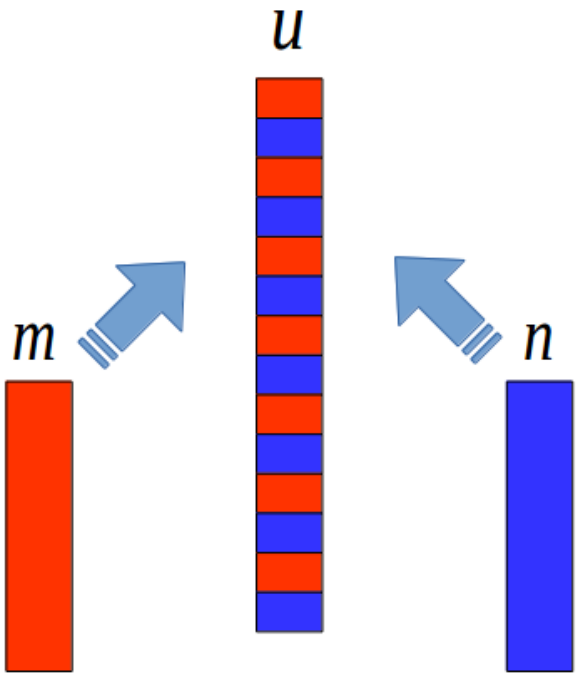


Figure 2.5: This illustrates how the two field, m and n, are weaved node after node in order to preserve the division by quadrants in parallel computing.

Chapter 3

Newton methods for nonlinear problems

Often in mathematical modeling nonlinear equations can be encountered and it is quite common that they can not be solved analytically. In this case, therefore, the solutions must be approached using *iterative methods*. In this chapter we will address the main aspects of theory of the Newton's method, which is a particular case of *fixed point iteration method*. This will be useful to prepare the ground for the numerical method that we have planned to test.

First of all we will introduce the standard Newton's method for nonlinear problems, in which each step solves a linear system where left-hand-side is the Jacobian of the nonlinear system and right-had-side the evaluation of the problem in a guess, that is computed from the previous step. Only the fist guess is chosen, and we will see the importance of this choice, since the convergence is local. Then we will list some of the most common *quasi Newton's methods*, that are methods in which the Jacobian is approximated, and indicate how the convergence is affected. On the other hand, also *quasi-Newton methods* will be introduced, that, at each step, solve the linear system non exactly. Our concern will be on Newton-Krylov methods and, in particular, on the Generalized Minimal Residual (GMRES). In section 3.2 we will give

an idea of why quasi-Newton methods and inexact Newton methods can be seen as equivalent.
In the end, we will illustrate some common techniques meant to globalizate the convergence, with more focus on line search approach.

1 Introduction of the method

Let us consider this nonlinear problem

$$\begin{cases} F(\mathbf{x}) = 0 \\ \mathbf{x} \in \mathbb{R}^n, \end{cases}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable.
Newton's method is an iterative method and its sequence is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - F'(\mathbf{x}_k)^{-1} F(\mathbf{x}_k), \tag{1.1}$$

where we start from a initial guess x_0. We can view (1.1) as the two-term Taylor expansion in which we impose $F(\mathbf{x}_{k+1}) = 0$. Therefore, at each step $\mathbf{x}_{k+1}$ is an approximation of the solution.

We introduce some assumptions and definitions useful for the convergence's theory.
Standard assumptions:

- $F(\mathbf{x}) = 0$ has a solution x^*;
- $F'(\mathbf{x}) : \Omega \to \mathbb{R}^n$ is Lipschitz continuous;
- $F'(\mathbf{x}^*)$ is non singular.

Convergence's definitions: *Let $\alpha \in \mathbb{R}^n$ and $\boldsymbol{x}_k \in \mathbb{R}^n$, $k = 0, 1, 2, ...$ Then $\boldsymbol{x}_k$ is said:*

- ***q-linearly convergent*** *if there exists a constant $C \in (0, 1)$ and an integer m such that for all $k \geq m$*

$$||\boldsymbol{x}_{k+1} - \alpha|| \leq C||\boldsymbol{x}_k - \alpha||.$$

- ***q-superlinearly convergent*** *if there exists a sequence C_k convergent to 0 such that*

$$||\boldsymbol{x}_{k+1} - \alpha|| \leq C_k ||\boldsymbol{x}_k - \alpha||.$$

- ***convergent sequence of q-order p*** $(p > 1)$ *if there exists a constant C and an integer $m > 0$ such that for all $k \geq m$*

$$||\boldsymbol{x}_{k+1} - \alpha|| \leq C ||\boldsymbol{x}_k - \alpha||^p.$$

Theorem 1.1. *Let the standard assumptions hold. Then there is a $\delta > 0$ such that, if the initial guess $\boldsymbol{x}_0 \in B(\delta)$, then the Newton iteration* (1.1) *converges q-quadratically to $\boldsymbol{x}^*$, that is $||\boldsymbol{e}_{k+1}|| \leq C||\boldsymbol{e}_k||$, for some $C > 0$ and with $||\boldsymbol{e}_k|| = ||\boldsymbol{x}_k - \boldsymbol{x}_{ex}||$.*

Since the initial guess needs to be "sufficiently near" to the solution, the convergence of the Newton's method is local.

A practical problem is that, in general, we do not have the analytical solution $\mathbf{x}_{ex}$, so we can not calculate the error $||\mathbf{e}_k|| = ||\mathbf{x}_k - \mathbf{x}_{ex}||$. Therefore, we have to find another estimation of the error that can be used as an *arrest criterion* of the iterate method. For example, it is used the *relative nonlinear residual* $||F(\mathbf{x})||/||F(\mathbf{x}_0)||$, that is a good indicator of size of the error, but only when $F'(\mathbf{x}^*)$ is well conditioned. Indeed we have:

Lemma 1. *Let the standard assumptions hold and $\delta > 0$ be small enough. Then for all $\boldsymbol{x} \in B(\delta)$*

$$\frac{||\mathbf{e}||}{4||\mathbf{e}_0||k(F'(\mathbf{x}^*))} \leq \frac{||F(\mathbf{x})||}{||F(\mathbf{x}_0)|} \leq \frac{4k(F'(\mathbf{x}^*))||\mathbf{e}||}{||\mathbf{e}_0||}$$

where $k(F'(\boldsymbol{x}^*)) = ||F'(\boldsymbol{x}^*)||\ ||F'(\boldsymbol{x}^*)^{-1}||$ is the condition number of $F'(\boldsymbol{x}^*)$ relative to the norm $||\cdot||$.

Another way to decide whether to terminate is to look at the Newton *step*

$$\mathbf{s}_{k+1} = \mathbf{x}_{k+1} - \mathbf{x}_k = -F'(\mathbf{x}_k)^{-1}F(\mathbf{x}_k),$$

and except $\mathbf{x}_{k+1}$ as good approximation of the solution when $||\mathbf{s}_{k+1}||$ is sufficiently small. This criterion is based on Theorem 1.1 which implies that

$$||\mathbf{e}_{k+1}|| = ||\mathbf{s}_{k+1}|| + \mathcal{O}(||\mathbf{e}_{k+1}||^2).$$

Hence, near the solution $\mathbf{s}$ and $\mathbf{e}$ are essentially the same size.

Sometimes it is too expensive from a computational point of view to evaluate $F'(\mathbf{x})$ at each step and sometimes, actually, there is no need to be so precise because, for example, we are too far from the solution. And also we would like to have a global convergence, instead just of a local one. There are many techniques that are done for these requirements, and in the next session we illustrate some of them.

2 Review of variants of the Newton method

This section will regard mainly different ways to approximate F'^{-1}, but first of all we want to give a theoretical result about inaccuracy. Suppose that $F + \epsilon$ and $F' + \Delta$ are used instead of F and F' in the iteration, then we have the following result.

Theorem 2.1. *Let the standard assumptions hold. Then there are $K > 0$, $\delta > 0$ and $\delta_1 > 0$ such that if $\boldsymbol{x}_k \in B(\delta)$ and $||\Delta(\boldsymbol{x}_k)|| < \delta_1$ then*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - (F'(\boldsymbol{x}_k) + \Delta(\boldsymbol{x}_k))^{-1}(F(\boldsymbol{x}_k) + \epsilon(\boldsymbol{x}_k))$$

is defined (i.e.,$F'(\boldsymbol{x}_k) + \Delta(\boldsymbol{x}_k)$ is nonsingular) and satisfies

$$||\boldsymbol{e}_{k+1}|| = K(||\boldsymbol{e}_k||^2 + ||\Delta(\boldsymbol{x}_n)||||\boldsymbol{e}_k|| + ||\epsilon(\boldsymbol{x}_k)||).$$

As we observe, it can happen that the convergence is not quadratical anymore.

2.1 Quasi-Newton methods

Under this name we have all the Newton methods that do not calculate the real value of $F'(\mathbf{x})^{-1}$, but use approximations. The price for such an approximation is that the nonlinear iteration converges more slowly; i.e., more nonlinear iterations are needed to solve the problem. However, the overall cost of solving is usually smaller, because the computation of the Newton step is less expensive. Therefore, we are obligated to use a quasi-Newton method when is unavailable to have the exact expression of $F'(\mathbf{x})^{-1}$ or is too expensive to compute it at every iteration.
Let us illustrate same of this methods.

Chord method or modified Newton method

In this case the Newton iteration is given by

$$\mathbf{x}_{k+1} = \mathbf{x}_k - F'(\mathbf{x}_0)^{-1} F(\mathbf{x}_k).$$

Suppose again that the standard assumptions hold. Assuming that the initial iteration is near enough to the solution $\mathbf{x}^*$, the convergence of the chord iteration is q-linear. Indeed, this comes from Theorem 2.1, noticing that $\epsilon(\mathbf{x}_k) = 0$ and $||\Delta(\mathbf{x}_k)|| = \mathcal{O}(||\mathbf{e}_0||)$.
In general, we can also update $F'(\mathbf{x})^{-1}$ after m iterations and not use $F'(\mathbf{x}_0)^{-1}$ always.
A general way to implement chord-type methods is

$$\mathbf{x}_{k+1} = \mathbf{x}_k - A^{-1} F(\mathbf{x}_k),$$

where $A \approx F'(\mathbf{x}^*)$. Also in this case, if we have a good guess $\mathbf{x}_0$ and A is a good approximation of $F'(\mathbf{x}^*)$, then we have a *q-linear* convergence.

Shamanskii method

It consists in a alternation of a Newton step with a sequence of chord steps and leads to a class of *high-order methods*, that is, methods that converge q-superlinearly with q-order larger that 2.
We can describe the transition from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$ by

$$\begin{aligned} \mathbf{y}_1 &= \mathbf{x}_k - F'(\mathbf{x}_k)^{-1} F(\mathbf{x}_k), \\ \mathbf{y}_{j+1} &= \mathbf{y}_j - F'(\mathbf{x}_k)^{-1} F(\mathbf{y}_j) \quad for\ 1 \leq j \leq m-1, \\ \mathbf{x}_{k+1} &= \mathbf{y}_m \end{aligned}$$

Note that for $m = 1$ it is Newton's method and for $m = \infty$ it is the simplest chord method.

Theorem 2.2. *Let the standard assumptions hold and let $m \geq 1$ be given. Then there are $K_S > 0$ and $\delta > 0$ such that if $\boldsymbol{x}_0 \in B(\delta)$, the Shamankii iterates converge q-superlinearly to $\boldsymbol{x}^*$ with q-order $m+1$ and*

$$||\boldsymbol{e}_{k+1}|| \leq K_S ||\boldsymbol{e}_k||^{m+1}.$$

The advantage of the Shamanskii method over Newton's method is that hight q-order can be optained with far fewer Jacobian evaluations or factorizations.

Difference approximations of the Jacobian matrix

Another possibility consists of replacing $F'(\mathbf{x}_k)$ with an approximation through n-dimensional differences of the form

$$(F_h'^k)_j = \frac{F(\mathbf{x}_k + h_j^k \mathbf{e}_j) - F(\mathbf{x}_k)}{h_j^k}, \quad \forall k \geq 0,$$

where $\mathbf{e}_j$ is the j-th vector of the canonical basis of $\mathbb{R}^n$ and $h_j^k > 0$ are increments to be suitably chosen at each step k of the iteration.

Convergence's result. Under the standards assumptions, a initial guess "sufficiently near" to the solution and bounded h_j^k for $j = 1, ..., n$, then the sequence

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [F_h'^k]^{-1} F(\mathbf{x}_k) \tag{2.1}$$

converges *linearly* to $\mathbf{x}^*$. Moreover, if there exists a positive constant C such that $\max_j |h_j^k| \leq C||\mathbf{x}_k - \mathbf{x}^*||$ or, equivalently, there exists a positive constant c such that $\max_j |h_j^k| \leq c||F(\mathbf{x}_k)||$, then the sequence (2.1), is convergent *quadratically.* But we should pay attention also not to choose h_j^k too small in order to avoid big errors of truncation.

Secant method

This case is done for single equations $f'(x) = 0$ and the derivative is approximated using a finite difference with the most recent two iterations.

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1}) f(x_k)}{f(x_k) - f(x_{k-1})} \tag{2.2}$$

For the computation of x_1, one option is to set $x_{-1} = 0.99x_0$. If the standard assumptions hold and x_{-1} and x_0 are sufficiently near the solution, Theorem 2.1, with $\epsilon = 0$ and $||\Delta(x_k)|| = \mathcal{O}(||e_{k-1}||)$, implies that the iteration converges *q-superlinearly.*

Broyden's method

This is a version of secant method for higher dimensions than 1.
In a multidimentional case the equation (2.2) has no sense, because we can not divide by a vector, so we ask that B_k, the current approximation od $F'(\mathbf{x})$, satisfies the secant equations

$$B_k(\mathbf{x}_k - \mathbf{x}_{k-1}) = F(\mathbf{x}_k) - F(\mathbf{x}_{k-1}).$$

A wide variety of methods, that satisfy the secant equations, have been designed to preserve such properties of the Jacobian as the sparsity patter

or symmetry. In the case of Broyden's method, we have

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \lambda_k B_k^{-1} F(\mathbf{x}_k).$$

where λ_k is the step length for Broyden direction

$$\mathbf{d}_k = -B_k^{-1} F(\mathbf{x}_k)$$

After the computation of $\mathbf{x}_{k+1}$, B_k is updated

$$B_{k+1} = B_k + \frac{(\mathbf{y} - B_k\mathbf{s})\mathbf{s}^T}{\mathbf{s}^T\mathbf{s}}$$

with $\mathbf{y} = F(\mathbf{x}_{k+1}) - F(\mathbf{x}_k)$ and $\mathbf{s} = \mathbf{x}_{k+1} - \mathbf{x}_k = \lambda_k \mathbf{d}_k$.
Broyden's method does not guarantee that the approximate Newton direction will be a descent direction for $||F||$ (the same can happen also with the secant method). Under hypothesis of standard assumptions and both $\mathbf{x}_0$ and B_0 are enough good approximations of $\mathbf{x}^*$ and $F'(\mathbf{x}^*)$, the convergence theory for Broyden's method is *q-superlinear*. So it is only local and, therefore, less satisfactory than that for the Newton and Newton-Krylov methods (we will see in the next subsections). Moreover the line search cannot be proved to compensate for poor initial iterate, because it can work for sure only with a good approximation of $F'(\mathbf{x}_k)$.

Approximation of a sparse Jacobian

In general the advantages of approximating J_k, the Jacobian matrix, instead of its inverse J_k^{-1}, are more evident when we have a sparse Jacobian with known nonzero positions. Indeed the inverse matrix could not be sparse and so it could need a storage much bigger and, if we know the pattern of the matrix, we can directly set these values to zero and have less conditions to solve. By the way, this scenario is typical for differential nonlinear problems, for which the Jacobian is sparse with known pattern.
We recall the general nonlinear problem $F(\mathbf{x}) = 0$ and its iterative step:

$$J_k \mathbf{s}_k = -F_k,$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda_k \mathbf{s}_k,$$

with λ_k a positive scalar, that reduces the length of the step to achieve stability, and J_k approximation of the Jacobian matrix. In order to find J_k, we use first primary conditions, that impose to J_k to predict the same changes along the direction $\mathbf{s}_k$ as F_k does, therefore

$$J_{k+1} = J_k + \frac{[F_{k+1} - (1-t_k)F_k]\mathbf{s}_k^T}{t_k \mathbf{s}_k^T \mathbf{s}_k}. \tag{2.3}$$

So we need to find conditions for the $n^2 - n$ unknowns that remain. These are called secondary conditions and they can be chosen differently, for example, J_k can be forced to give the same variations that F_k has along all orthogonal directions on $\mathbf{s}_k$.

As we can see, the logic is the one used Broyden's method, but, here, it has to be adapted to the sparse case, below we see how.

The ith row $g_k^{(i)}$ of J_k is an approximation to the gradient of the ith function component $F_k^{(i)}$. If $n - r_i$ components of $g_k^{(i)}$ are known constants, we impose first the condition that these components shall remain unchanged in the Jacobian revision, and then we implement the other conditions on the basis of the remaining r_i coordinate directions. The vectors $\hat{\mathbf{s}}_k$ and $\bar{g}^{(i)}$ are introduced, where the first one is a column vector that is the same as $\mathbf{s}_k$, but setting $\mathbf{s}_k^{(j)}$ to zero whenever the corresponding element of $g_k^{(i)}$ is a known constant. Indeed, $\bar{g}^{(i)}$ is a row vector that is as $g_k^{(i)}$, but setting its unknown elements to zero. The primary conditions becomes

$$t_k g_{k+1}^{(i)} \hat{\mathbf{s}}_k = F_{k+1}^{(i)} - F_k^{(i)} - t_k \bar{g}^{(i)} \mathbf{s}_k, \quad i = 1, 2, \ldots, n. \tag{2.4}$$

This is identical to the primary condition $t_k J_{k+1} \mathbf{s}_k = F_{k+1} - F_k$, because $g_{k+1}^{(i)} \hat{\mathbf{s}}_k + \bar{g}^{(i)} \mathbf{s}_k = g_{k+1}^{(i)} \mathbf{s}_k$.

The secondary condition is obtained in a similar way by restricting the previous secondary condition to the r_i-space corresponding to the unknown elements of g_i, that is:

$$g_{k+1}^{(i)} \hat{q} = g_k^{(i)} \hat{q}, \quad i = 1, 2, \ldots, n, \tag{2.5}$$

where $\hat{q}$ satisfies $\hat{\mathbf{s}}_k^T \hat{q} = 0$. It can be verified that (2.4) and (2.5) are satisfied by the exact row-by-row analogue of (2.3), that is

$$g_{k+1}^{(i)} = g_k^{(i)} + \frac{[F_{k+1}^{(i)} - (1 - t_k)F_k^{(i)}]\hat{\mathbf{s}}_k^T}{t_k \hat{\mathbf{s}}_k^T \hat{\mathbf{s}}_k}.$$

All this method is stated in [14] by L.K. Schubert and it is also shown that, when the dimension n increases, this approach is much better than classical Broyden's method for a sparse Jacobian.

3 Inexact Newton method

Rather than approximate the Jacobian, one could instead solve the equation for the Newton step approximately. An inexact Newton method uses as a step a vector $\mathbf{s}$ that satisfies the inexact Newton condition

$$||F'(\mathbf{x}_k)\mathbf{s} + F(\mathbf{x}_k)|| \leq \eta ||F(\mathbf{x}_k)||. \tag{3.1}$$

The parameter η is called **forcing term**. Away from the solution $\mathbf{x}^*$, F and its local linear model may disagree considerably at a step that closely approximates the Newton step. So choosing η too small may lead to *oversolving* the Newton equation. Therefore far from the solution, a less accurate approximation of the Newton step may be both cheaper and more effective. So, the idea is to choose a forcing term that becomes smaller when the iteration are closer to the solution. And what about the convergence?

Theorem 3.1. *Let the standard assumptions hold. Then there are δ and $\bar{\eta}$ such that, if $\boldsymbol{x}_0 \in B(\delta)$, $\eta_n \subset [0, \bar{\eta}]$, then the inexact Newton iteration*

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$$

where

$$||F'(\boldsymbol{x}_k)\boldsymbol{s}_k + F(\boldsymbol{x}_k)|| \leq \eta_k ||F(\boldsymbol{x}_k)||,$$

converges q-linearly to $\boldsymbol{x}^*$. Moreover,

- *if* $\eta_n \to 0$*, the convergence is q-superlinear, and*
- *if* $\eta_k \leq K_\eta ||F(\boldsymbol{x}_k)||^p$ *for some* $K_\eta > 0$*, the convergence is q-superlinear with q-order* $1 + p$*.*

In literature, in particular [6], we can find the following choices of η.

Choice 1. Given $\eta_0 \in [0, 1)$, choose

$$\eta_k = \frac{||F(\mathbf{x}_k) - F(\mathbf{x}_{k-1}) - F'(\mathbf{x}_{k-1})\mathbf{s}_{k-1}||}{||F(\mathbf{x}_{k-1})||}, \qquad k = 1, 2, 3... \tag{3.2}$$

or

$$\eta_k = \frac{|||F(\mathbf{x}_k)|| - ||F(\mathbf{x}_{k-1}) + F'(\mathbf{x}_{k-1})\mathbf{s}_{k-1}|||}{||F(\mathbf{x}_{k-1})||}, \qquad k = 1, 2, 3... \tag{3.3}$$

Note that η_k given by (3.2) and (3.3) directly reflects the agreement between F and its local linear model at the previous step. The choice (3.3) may be more convenient to evaluate that (3.2) in some circumstances. Since it is at least as small as (3.2), local convergence will be at least as fast as with (3.2).

One possible way to obtain faster local convergence, while retaining the potential advantages of (3.2) and (3.3), is to raise those expression to powers greater than one.

Choice 2. Given $\gamma \in [0, 1]$, $\alpha \in (1, 2]$, and $\eta_0 \in [0, 1)$, choose

$$\eta_k = \gamma \left(\frac{||F(\mathbf{x}_k)||}{||F(\mathbf{x}_{k-1})||} \right)^\alpha, \qquad k = 1, 2, 3... \tag{3.4}$$

This choice does not directly reflect the agreement between F and his local linear model. However, it can produce a little oversolving in practice.

In experiments it was observed that the forcing term choices occasionally become too small far away from a solution. There is a particular danger of the Choice 1 forcing terms becoming too small; indeed, a η_k given by (3.2) or (3.3) can be undesirably small because of their very small step or coincidental very good agreement between F and its local linear model. For this reason there are **safegurads** that are intended to prevent the forcing

term to become too small too soon.
Choice 1 safeguard: Modify η_k by $\eta_k = max\{\eta_k, \eta_{k-1}^{\frac{(1+\sqrt{5})}{2}}\}$ whenever $\eta_{k-1}^{\frac{(1+\sqrt{5})}{2}} > 0.1$.
Choice 2 safeguard: Modify η_k by $\eta_k = max\{\eta_k, \gamma\eta_{k-1}^{\alpha}\}$ whenever $\gamma\eta_{k-1}^{\alpha} > 0.1$.
As we can see, they are activated when the previous η was not so small as the new one tries to be, so it is more suspicious.
There is also another version of safeguard for the choice 2 in [10], and this is

$$\eta_k = min(\eta_{max}, max(\eta_k^C, 0.5\tau_t/||F(\mathbf{x}_k)||)),$$

with $\tau_t = \tau_a + \tau_r||F(\mathbf{x}_0)||$ and

$$\eta_k^C = \begin{cases} \eta_{max}, & k = 0 \\ min(\eta_{max}, \eta_k), & k > 0, \quad \gamma\eta_{k-1}^2 \leq 0.1 \\ min(\eta_{max}, max(\eta_k, \gamma\eta_{k-1}^2)), & k > 0, \quad \gamma\eta_{k-1}^2 > 0.1 \end{cases}$$

Iterative methods for solving the equation for the Newton step would typically use (3.1) as a termination criterion. In this case, the overall nonlinear solver is called a **Newton iterative method**, and they are named by the particular iterative method used for the linear equation. For example, there are Newton-Jacobi, Newton-SOR or Newton-Krylov.

3.1 Newton-Krylov methods

As we said before, for each iteration of the inexact Newton method, we have to solve a linear equation with an iterative method. Sometimes we refer to this linear iteration as an *inner iteration*. Similarly, the nonlinear iteration is ofen called the *outer iteration*.
The Newton-Krylov methods, as the name suggests, use Krylov subspace-based linear solver. It approximates the solution of a linear system $A\mathbf{d} = \mathbf{b}$

with a sum of the form

$$\mathbf{d}_k = \mathbf{d}_0 + \sum_{j=0}^{k-1} \gamma_j A^j \mathbf{r}_0,$$

where $\mathbf{r}_0 = \mathbf{b} - A\mathbf{d}_0$ and $\mathbf{d}_0$ is the initial iterate. Since the goal is to approximate a Newton step, the most sensible initial iterate is $\mathbf{d}_0 = 0$, because we have no priory knowledge of the direction, but, at least in the local phase of the iteration, expect it ot be small.
We express this in compact form as $\mathbf{d}_k \in \mathcal{K}_k$, where the kth **Krylov subspace** is $\mathcal{K}_k = span(\mathbf{r}_0, A\mathbf{r}_0, ..., A^{k-1}\mathbf{r}_0)$.
There are many Newton-Krylov methods and they differ in storage requirements, cost in evaluations of F and robustness. If A is symmetric and positive definite, the conjugate gradient (CG) method has better storage and convergence properties than others Newton-Krylov methods. If the matrix A does not have this properties, then we can use two low-storage solvers, BiCGSTAB and TFQMR, but we have to be aware that this solvers can break down, because a division by zero can occur. An other option is GMRES, that is not low-storage solver (but it can become as we will see in the next section), and, when there is no convergence, instead of breaking down, there is just a stagnation in the iterations.

GMRES

The kth Generalized Minimal Residual (GMRES) iterate is the solution of the linear least squares problem of minimizing

$$||\mathbf{b} - A\mathbf{d}_k||^2$$

over $\mathcal{K}_k$.
An important property of the method, it that GMRES must accumulate the history of the linear iteration as an orthonormal basis for the Krylov subspaces. Therefore, for large problems, it can exhaust the available fast memory. For these cases, there is GMRES(m), which restarts the iteration

when the size of Krylov space exceeds m vectors.
Sometimes GMRES method, like other Krylov methods, is implemented as a *matrix-free*. The reason is that only matrix-vector products, rather than details of the matrix, are needed to implement a Krylov method.
Algorithm. We want to solve the linear system

$$A\mathbf{d} = \mathbf{b},$$

using the l_2-orthogonal basis $V_k = \{\mathbf{v}_1, ..., \mathbf{v}_k\}$ of the space $\mathcal{K}_k$; in fact, we look for a solution of the type $\mathbf{d}_k = \mathbf{d}_0 + \mathbf{z}_k$, where $\mathbf{d}_0$ is an initial guess and $\mathbf{z}_k \in \mathcal{K}_k$. Let us define the residual $\mathbf{r}_k = \mathbf{b} - A\mathbf{d}_k$ and choose $\mathbf{v}_1 = \frac{\mathbf{r}_0}{||\mathbf{r}_0||}$. We can write $\mathbf{z}_k$ as a linear combination of $\{\mathbf{v}_1, ..., \mathbf{v}_k\}$, that is

$$\mathbf{z}_k = V_k \mathbf{y}_k.$$

We introduce H, the upper $k \times k$ Hessenberg matrix, that is $H \equiv V_k^T A V_k$. It is the matrix representation of A_k in the basis $\{\mathbf{v}_1, ..., \mathbf{v}_k\}$, where A_k is the l_2-orthogonal projection of A in $\mathcal{K}_k$.
Since we have $(\mathbf{b} - A(\mathbf{d}_k)) \perp \mathcal{K}_k$, that is $(\mathbf{b} - A(\mathbf{d}_0 + \mathbf{z}_k)) \perp \mathcal{K}_k$, then we know that $(\mathbf{w}, \mathbf{r}_0 - A\mathbf{z}_k) = 0 \ \ \forall \mathbf{w} \in \mathcal{K}_k$. We can deduce that $V_k^T A \mathbf{z}_k = V_k^T \mathbf{r}_0$, and so $A\mathbf{z}_k = \mathbf{r}_0$ for $k = n$, dimension of the system. That means that $A\mathbf{d}_n = \mathbf{b}$. It is known that $\mathbf{r}_0 = V_k \mathbf{e}_1 ||\mathbf{r}_0||$, with $\mathbf{e}_1$ the unit vector $\mathbf{e}_1 = (1, 0, ..., 0)^T$, then we deduce that $\mathbf{y}_k = H_k^{-1} ||\mathbf{r}_0|| \mathbf{e}_1$.

ALGORITHM 1 (*Arnoldi's method*): Full orthogonalization method.

1. *Start:* Choose $\mathbf{d}_0$ and compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{d}_0$ and $v_1 = \frac{\mathbf{r}_0}{||\mathbf{r}_0||}$.
2. *Iterate:* For $j = 1, 2, ..., k$ do:
   $h_{i,j} = (A\mathbf{v}_j, \mathbf{v}_i), \ \ i = 1, 2, ...j,$
   $\hat{\mathbf{v}}_{j+1} = A\mathbf{v}_j - \sum_{i=1}^{j} h_{i,j} \mathbf{v}_i,$
   $h_{j+1,j} = ||\hat{\mathbf{v}}_{j+1}||,$
   $\mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1} / h_{j+1,j}.$
3. *Form the solution:*
   $\mathbf{d}_k = \mathbf{d}_0 + V_k \mathbf{y}_k$ where $\mathbf{y}_k = H_k^{-1} ||\mathbf{r}_0|| \mathbf{e}_1$

The step 2 of the ALGORITHM 1 just uses the Gram-Schmidt method for computing an l_2-orthonormal basis $\{\mathbf{v}_1, ..., \mathbf{v}_k\}$.

We see now the GMRES algorithm that comes from the Arnoldi's one. After k steps of Arnoldi's method, we have an l_2-orthonormal system V_{k+1} and a $(k+1) \times k$ matrix $\bar{H}_k$, whose only non zero entries are the element h_{ij} generated by the method. Thus the $\bar{H}_k$ is the same as H_k except for an additional row, whose only nonzero element is $h_{k+1,k}$. We have this important relation:

$$AV_k = V_{k+1}\bar{H}_k. \tag{3.5}$$

We would like to solve the least squares problem:

$$\min_{\mathbf{z}\in\mathcal{K}_k} ||\mathbf{f} - A[\mathbf{d}_0 + \mathbf{z}]|| = \min_{\mathbf{z}\in\mathcal{K}_k} ||\mathbf{r}_0 - A\mathbf{z}||. \tag{3.6}$$

We remember that $\mathbf{z} = V_k\mathbf{y}$, so we can see (3.6) as a function of $\mathbf{y}$ to be minimized:

$$J(\mathbf{y}) = ||\beta\mathbf{v}_1 - AV_k\mathbf{y}||,$$

where we have $\beta = ||\mathbf{r}_0||$. Using 3.5 we obtain

$$J(\mathbf{y}) = ||V_{k+1}(\beta\mathbf{v}_1 - \bar{H}_kV_k\mathbf{y})||.$$

Recalling the fact that V_{k+1} is l_2-orthonormal and so that it preserve the norm, we see that

$$J(\mathbf{y}) = ||\beta\mathbf{v}_1 - \bar{H}_kV_k\mathbf{y}||. \tag{3.7}$$

In conclusion, the solution of the least squares problem (3.6) is given by

$$\mathbf{d}_k = \mathbf{d}_0 + V_k\mathbf{y}_k,$$

where $\mathbf{y}_k$ minimizes the function $J(\mathbf{y})$, defined by (3.7), over $\mathbf{y} \in \mathbb{R}^k$.

ALGORITHM 2 (*GMRES*): Full orthogonalization method.

1. *Start:* Choose $\mathbf{d}_0$ and compute $\mathbf{r}_0 = \mathbf{b} - A\mathbf{d}_0$ and $\mathbf{v}_1 = \frac{\mathbf{r}_0}{||\mathbf{r}_0||}$.
2. *Iterate:* For $j = 1, 2, ..., k$ do:

$h_{i,j} = (A\mathbf{v}_j, \mathbf{v}_i), \quad i = 1, 2, ...j,$
$\hat{\mathbf{v}}_{j+1} = A\mathbf{v}_j - \sum_{i=1}^{j} h_{i,j}\mathbf{v}_i,$
$h_{j+1,j} = ||\hat{\mathbf{v}}_{j+1}||,$
$\mathbf{v}_{j+1} = \hat{\mathbf{v}}_{j+1}/h_{j+1,j}.$

3. *Form the approximate solution:*

$\mathbf{d}_k = \mathbf{d}_0 + V_k\mathbf{y}_k$ where $\mathbf{y}_k$ minimizes (3.7)

How to compute the step 3 of ALGORITHM 2 practically ?
We consider the QR-factorization of $\bar{H}_k$, so $Q_k\bar{H}_k = R_k$, with Q_k a $(k+1) \times (k+1)$ rotation matrix and R_k a $(k+1) \times k$ upper triangular matrix whose last row is zero. Since Q_k is unitary, we have:

$$J(\mathbf{y}) = ||\beta\mathbf{e}_1 - \bar{H}_k\mathbf{y}|| = ||Q_k(\beta\mathbf{e}_1 - \bar{H}_k\mathbf{y})|| = ||\mathbf{g}_k - R_k\mathbf{y}||, \tag{3.8}$$

where $\mathbf{g}_k = Q_k\beta\mathbf{e}_1$. Since the last row of R_k is zero, the minimization of (3.8) is achieved by solving the upper triangular linear system which we have if we remove the last row of R_k and the last component of $\mathbf{g}_k$. We also notice that the residual norm of the solution $\mathbf{d}_k$ is equal to the $(k+1)$st component of $\mathbf{g}_k$.
For more details see [13].

Convergence. As a general rule, GMRES, like others Krylov methods, performs best if the eigenvalues of A are in a few tight clusters. One way to see this, keeping in mind $\mathbf{d}_0 = 0$, is to observe the kth GMRES residual can be written as a polynomial in A applied to the residual

$$\mathbf{r}_k = \mathbf{b} - A\mathbf{d}_k = p(A)\mathbf{r}_0 = p(A)\mathbf{b}.$$

Here $p \in \mathcal{P}_k$, this is the set of polynomial of degree k with $p(0) = 1$. Since the kth GMRES iteration satisfies

$$||A\mathbf{d}_k - \mathbf{b}|| \leq ||A\mathbf{z} - \mathbf{b}||$$

for all $\mathbf{z} \in K_k$, we must have

$$||\mathbf{r}_k|| = \min_{p \in \mathcal{P}_k} ||p(A)\mathbf{r}_0||.$$

This simple fact can lead to very useful error estimates. Suppose A is diagonalizable, in other words there is a nonsingular matrix V such that

$$A = V\Lambda V^{-1}.$$

Here Λ is a diagonal matrix with the eigenvalues of A on the diagonal. If A ia a diagonalizable matrix and p is a polynomial, then

$$p(A) = Vp(\Lambda)V^{-1}.$$

Theorem 3.2. *Let $A = V\Lambda V^{-1}$ be a nonsingular diagonalizable matrix. Let $\boldsymbol{d}_k$ be the kth GMRES iterate. Then, for all $\bar{p_k} \in \mathcal{P}_k$,*

$$\frac{||\boldsymbol{r}_k||}{||\boldsymbol{r}_0||} \leq k(V) \max_{\boldsymbol{z}\in\sigma(A)} |\bar{p_k}(\boldsymbol{z})|.$$

Suppose, for example, that A is diagonalizable, $k(V) = 100$, and all the eigenvalues of A lies in a disk of radius 0.1 centered about 1 in the complex plane. Theorem 3.2 implies (using $\bar{p_k}(\mathbf{z}) = (1 - \mathbf{z})^k$) that

$$||\mathbf{r}_k|| \leq 100(0.1)^k = 0.1^{k-2}$$

Hence, GMRES will reduce the residual by a factor of, say, 10^5 after seven iterations. And now we can see also that having clusters that are not so spread, is better. One aim of preconditioning is to change A to obtain an advantageous distribution of eigenvalues.

3.2 Equivalences between inexact and Quasi-Newton method

Consider the quasi-Newton iterate:

$$(F'(\mathbf{x}_k) + \Delta_k)\mathbf{s}_k = -F(\mathbf{x}_k),$$

with $\Delta_k = B_k - F'(\mathbf{x}_k)$ and B_k a sequence of invertible matrices. From Theorem 1 in [5], we know that, if QN iterates converge to $\mathbf{x}^*$, then they converge *superlinearly* if and only if

$$\frac{||(B_k - F'(\mathbf{x}^*))(\mathbf{x}_{k+1} - \mathbf{x}_k)||}{||\mathbf{x}_{k+1} - \mathbf{x}_k||} \to 0 \;\; as\; k \to \infty \tag{3.9}$$

The last one can be written also like this:

$$||(F'(\mathbf{x}_k) + \Delta_k - F'(\mathbf{x}^*))\mathbf{s}_k|| = o(||\mathbf{s}_k||) \ \ as\ k \to \infty \tag{3.10}$$

Now, let consider the inexact Newton iterate:

$$F(\mathbf{x}_k)s_k = -F(\mathbf{x}_k) + \mathbf{r}_k$$

The Theorem 2 in [5] asserts that if the IN iterates converge to $\mathbf{x}^*$, then this convergence is *superlinear* if and only if

$$||\mathbf{r}_k|| = o(||F(\mathbf{x}_k)||) \ \ as\ k \to \infty \tag{3.11}$$

We will call (3.9) and (3.11) conditions that characterize the superlinear convergence. Now we write the QN iterates as IN iterates is this way

$$F'(\mathbf{x}_k)\mathbf{s}_k = -F(\mathbf{x}_k) - \Delta_k\mathbf{s}_k,$$

and in this case (3.11) becomes

$$||\Delta_k\mathbf{s}_k|| = o(||F(\mathbf{x}_k)||) \ \ as\ k \to \infty. \tag{3.12}$$

In [5] it is proved that (3.12) and (3.10) are equivalent.

Moreover, also the IN iterates can be written as QN iterates

$$(F'(\mathbf{x}_k) - \frac{1}{||\mathbf{s}_k||_2^2}\mathbf{r}_k\mathbf{s}_k^t)\mathbf{s}_k = -F(\mathbf{x}_k)$$

and (3.10) becomes

$$||(F'(\mathbf{x}_k) - \frac{1}{||\mathbf{s}_k||_2^2}\mathbf{r}_k\mathbf{s}_k^t - F'(\mathbf{x}^*))\mathbf{s}_k|| = o(||\mathbf{s}_k||) \ \ as\ k \to \infty \tag{3.13}$$

Also in this case, in [5] is proven that (3.13) and (3.11) are equivalent.

The conclusion is that quasi-Newton methods and the inexact Newton methods are equivalent, in the sense that each may be used to characterize the high convergence order of the other. For example, one can use Theorem 3.1 to analyze the chord method or the secant method. In the case of the chord method, the steps satisfy (3.1) with $\eta_k = \mathcal{O}(||\mathbf{e}_0||)$, which implies q-linear convergence if $||\mathbf{e}_0||$ is sufficiently small. For the secant method, $\eta_k = \mathcal{O}(||\mathbf{e}_{k-1}||)$, implying q-superlinear convergence.

4 Global convergence

The convergence of Newton and inexact Newton methods is local; i.e., convergence is guaranteed if the initial iterate $\mathbf{x}_0$ is sufficiently near a solution. Globalization techniques improve the likelihood of convergence from arbitrary starting points and most of them fall into two classes: **line search** methods and **trust-region** methods.

4.1 Line search

Many times, when we are far from the solution, we do not manage to arrive to convergence, because the steps become too large or too small. In this case, with an extra condition, we decide how large to be the step in the **search direction** $\mathbf{d}_k$ calculated in that iteration. So we do not update $\mathbf{x}_{k+1}$ anymore in this way $\mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{d}_k$, but we do as follows: $\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda \mathbf{d}_k$. Line search with NewtonâĂŹs method is called also **damped Newton method**. One of the simplest condition imposes that the new $\mathbf{x}_{k+1}$ has to make *decrease* $||F||$, therefore:

$$\begin{gathered} \lambda = 1 \\ \text{while} \quad ||F(\mathbf{x}_k + \lambda \mathbf{d}_k)|| \geq ||F(\mathbf{x})|| \\ \lambda = \lambda/2 \\ \text{endwhile} \\ \mathbf{x}_{k+1} = \mathbf{x}_k + \lambda \mathbf{d}_k \end{gathered}$$

We call this method *line search* because we search along the line segment

$$(\mathbf{x}_k, \mathbf{x}_k - \mathbf{d}_k)$$

to find a decrease of $||F||$. As we move from the right endpoint to the left, usually this procedure is called *backtracking*.
There is also another condition, that impose the new $\mathbf{x}_{k+1}$ to make *sufficiently*

decrease $||F||$,

$$\lambda = 1 \text{ and } \alpha \in (0,1)$$
$$\text{while} \quad ||F(\mathbf{x}_k + \lambda \mathbf{d}_k)|| \geq (1 - \alpha\lambda)||F(\mathbf{x})||$$
$$\lambda = \lambda/2$$
$$\text{endwhile}$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + \lambda \mathbf{d}_k$$

This approach is called **Armijo rule**.

In these cases the factor of reduction is $\frac{1}{2}$, but sometimes a factor of 10 could be better if small values of λ are needed for several consecutive steps. On the other hand, reducing λ too much can be costly as well. Taking full Newton steps ensures fast local convergence. Taking *as large* a fraction *as possible* helps to move the iteration into the terminal phase in which full steps may be taken and fast convergence expected.

There are different ways of choosing λ.

Constant reduction. As we saw, one possibility is to halve λ at each try, but we can also choose other ways, like, for example, choosing a starting $\lambda_0 \in (0,1)$ and then use for each try m , $\lambda = \lambda_0^m$.

Polynomial line searches. Choosing a reduction factor of the steplegngth that is the best for the specific step is better than constant reduction factors. If one can model the decrease in $||F||$ as the steplength is reduced, one might expect to be able to better estimate the appropriate reduction factor. In practice such methods usually perform better than constant reduction factors.

If we have rejected k steps, we have in hand the values

$$||F(\mathbf{x}_k)||, ||F(\mathbf{x}_k + \lambda_1 \mathbf{d}_k)||, ..., ||F(\mathbf{x}_k + \lambda_{k-1} \mathbf{d}_k)||.$$

We can use this iteration history to model the scalar function

$$f(\lambda) = ||F(\mathbf{x}_k + \lambda \mathbf{d}_k)||^2$$

with a polynomial and use the minimum of that polynomial as the next steplength. We consider two ways that use second degree polynomials which

we compute using previously computed information. After λ_c has been rejected and a model polynomial computed, we compute the minimum λ_t of that polynomial analytically and set

$$\lambda_+ = \begin{cases} \sigma_0\lambda_c \;\; if \;\; \lambda_t < \sigma_0\lambda_c, \\ \sigma_1\lambda_c \;\; if \;\; \lambda_t > \sigma_1\lambda_c, \\ \lambda_t \quad \text{otherwise} \end{cases} \tag{4.1}$$

with σ_0 and σ_1 being safeguards avoiding to have λ too small or large. In general, these safeguards are set to 0.1 and 0.5.

Let's see how to calculate λ_t.

Two-point parabolic model. Here we use values of $f(0)$ and $f'(0)$ and the value of f at the current λ to construct a 2nd degree interpolating polynomial for f.

We have $f(0) = ||F(\mathbf{x}_k)||^2$ and $f'(0) = 2F(\mathbf{x}_k)^T(F'(\mathbf{x}_k)\mathbf{d}_k)$, where $(F'(\mathbf{x}_k)\mathbf{d}_k)$ can be obtained by examination of the final residual of GMRES. Moreover, $f'(0)$ needs to be negative and if it does not happen, then we may need a new search direction.

Therefore the polynomial is:

$$p(\lambda) = f(0) + f'(0)\lambda + c\lambda^2$$

with

$$c = \frac{f(\lambda_c) - f(0) - f'(0)\lambda_c}{\lambda_c^2}$$

Having $f'(0) < 0$, so if $f(\lambda_c) > f(0)$, then $c > 0$ and $p(\lambda)$ has a minimum at

$$\lambda_t = -f'(0)/(2c) > 0.$$

We then compute λ_+ with (4.1).

Three-point parabolic model. An alternative to the two-point model, that avoids the need to approximate $f'(0)$, is a three-point model, which uses $f(0)$ and the two most recently rejected steps to create the parabola.

In this case we evaluates $f(0)$ and $f(1)$ and, if the full step is rejected, we

set $\lambda = \sigma_1$ and try again. After rejection, we have the values

$$f(0),\ f(\lambda_c)\ \ and\ \ f(\lambda_p)$$

where λ_c and λ_p are the most recently rejected values of λ. The polynomial that interpolates f at 0, λ_c, λ_p is

$$p(\lambda) = f(0) + \frac{\lambda}{\lambda_c - \lambda_p}\left(\frac{(\lambda - \lambda_p)(f(\lambda_c) - f(0))}{\lambda_c} + \frac{(\lambda_c - \lambda)(f(\lambda_p) - f(0))}{\lambda_p}\right)$$

We must consider two situations. If $p''(0) > 0$, then we set λ_t to the minimum of p, $\lambda_t = -p'(0)/p''(0)$, and apply safeguarding (4.1) to compute λ_+. If $p''(0) \leq 0$ one could either set λ_+ to be minimum of p on the interval $[\sigma_0\lambda, \sigma_1\lambda]$ or reject the parabolic model and simply set $\lambda_+ = \sigma_1\lambda_c$.

4.2 Globalization in inexact Newton method

As we saw, the globalization of the convergence in Newton methods is about ensuring that the step $k+1$ will reduce the norm of F, even through a constant less than 1, that is $||F(\mathbf{x}_k + 1)|| < \alpha||F(\mathbf{x}_k)||$, with $0 < \alpha \leq 1$. That means the direction $\mathbf{s}_k$ should be a *descent direction.*

Consider this minimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^N} f(\mathbf{x}) = \frac{1}{2}F(\mathbf{x})^T F(\mathbf{x}).$$

A descent direction for f at the current approximation $\mathbf{x}$ is any vector $\mathbf{p}$ such that:

$$\nabla f(\mathbf{x})^T \mathbf{p} < 0. \tag{4.2}$$

Easily we can show that, being $\nabla f(\mathbf{x})^T \mathbf{p} = J(\mathbf{x})^T F(\mathbf{x})$, with $J(\mathbf{x}) = F'(\mathbf{x})$, (4.2) is equal to

$$F(\mathbf{x})^T J(\mathbf{x})\mathbf{p} < 0. \tag{4.3}$$

For such a direction, it is shown that there exists a certain $\lambda_0 > 0$ such that $f(\mathbf{x} + \lambda\mathbf{p}) < f(\mathbf{x})\ \ \forall\lambda\ :\ 0 < \lambda \leq \lambda_0$.

If we are solving $J(\mathbf{x}_k)\mathbf{s}_k = -F(\mathbf{x}_k)$ with a *direct method*, so $\mathbf{s}_k$ is the exact solution, because of the characteristics of Newton methods, $\mathbf{s}_k$ will be always a descent direction; indeed, il we put $\mathbf{p} = -J(\mathbf{x})^{-1}F(\mathbf{x})$, (4.3) is verified. This is not always true when we just approximate $\mathbf{s}_k$ with an *iterative method*.

Consider to be in the case of Newton-Krylov method with GMRES and put for simplicity $F = F(\mathbf{x})$ and $J = J(\mathbf{x})$, consider $\bar{\mathbf{s}}$ as an approximation of $J\mathbf{s} = -F$. Then we can write

$$F^T J\bar{\mathbf{s}} = -F^T F - F^T\bar{\mathbf{r}}$$

with $\bar{\mathbf{r}} = -F - J\bar{\mathbf{s}}$; $\bar{\mathbf{s}}$ will be the descent direction for f at $\mathbf{x}$ whenever $|F^T\bar{\mathbf{r}}| < F^T F$, in particular, if

$$||\bar{\mathbf{r}}|| < ||F||, \tag{4.4}$$

then $\bar{\mathbf{d}}$ is a descent direction.

Condition (4.4) means that the norm in GMRES must be reduced strictly. It holds whenever the Jacobian matrix J is positive real and at least one step of GMRES is performed. But these hypothesis on J are too restrictive, so a milder condition is to assume that the dimension m in GMRES is large enough to ensure that the final residual is reduced by a factor of at least η, where η is a scalar less than 1. This is how we arrive again at the expression of inexact Newton method

$$||F'(\mathbf{x}_k)\mathbf{s} + F(\mathbf{x}_k)|| \leq \eta||F(\mathbf{x}_k)||.$$

Trust region

Let $\mathbf{x}$ be the current approximate solution of $F(\mathbf{x}) = 0$. The effect of using a Krylov method to solve the Newton equations $J(\mathbf{x})\mathbf{d} = -F(\mathbf{x})$ approximately is to take a step from $\mathbf{x}$ of the form $\mathbf{x}+\mathbf{s}$, where $\mathbf{s}$ is in the affine

subspace $\mathbf{d}_0 + \mathcal{K}_m$. If $V_m = [\mathbf{v}_1, ..., \mathbf{v}_m]$ is an orthonormal basis for $\mathcal{K}_m$ and the initial guess $\mathbf{d}_0 = 0$, then $\mathbf{d} = V_m\mathbf{y}$, for some $\mathbf{y} \in \mathbb{R}^m$, and we have a step of the form $\mathbf{x} + V_m\mathbf{y}$.

As we introduced before, our global strategy will again be based upon finding a local minimum of the real-valued function $\frac{1}{2}F(\mathbf{x})^T F(\mathbf{x})$. Thus, we want to solve

$$\min_{\mathbf{y}\in\mathbb{R}^m} f(\mathbf{x} + V_m\mathbf{y}). \tag{4.5}$$

Letting $g(\mathbf{y}) = f(\mathbf{x} + V_m\mathbf{y})$, we have

$$\nabla g(\mathbf{y}) = (J(\mathbf{x} + V_m\mathbf{y})V_m)^T F(\mathbf{x} + V_m\mathbf{y})$$

and, in particular, that

$$\nabla g(0) = (JV_m)^T F.$$

If we use $F + JV_m\mathbf{y}$ as a linear model of $F(\mathbf{x} + V_m\mathbf{y})$, then the quadratic model for g is

$$\hat{g}(\mathbf{y}) = \frac{1}{2}||F + JV_m\mathbf{y}||^2$$

Letting $B_m = V_m^T J^T JV_m$, we have

$$\hat{g}(\mathbf{y}) = \frac{1}{2}F^T F + F^T JV_m\mathbf{y} + \frac{1}{2}\mathbf{y}^T B_m\mathbf{y} \tag{4.6}$$

where B_m is symmetric and positive semidefinite, and $\nabla\hat{g}(0) = \nabla g(0)$. If J is non-singular, then B_m is positive definite, since V_m has orthonormal columns. The model trust region approach, that we are considering, will be based upon trying to find a solution of the problem

$$\min_{||\mathbf{y}||\leq\tau} \hat{g}(\mathbf{y}), \quad \mathbf{y} \in \mathbb{R}^m \tag{4.7}$$

where τ is an estimate of the maximum length of a successful step to take from $\mathbf{x}$ and, also, a measure of the size of the region in which the local quadratic model $\hat{g}(\mathbf{y})$ closely agrees with the function $g(\mathbf{y})$. The solution of (4.7) is in this theorem:

Theorem 4.1. *Let $\hat{g}(\boldsymbol{y})$ be defined by* (4.6)*, and assume that J is nonsingular. Then problem* (4.7) *is solved by*

$$\boldsymbol{y}_m(\mu) = (B_m + \mu I)^{-1} \boldsymbol{z}_m,$$

where $\boldsymbol{z}_m = -\nabla \hat{g}(0)$, for the unique μ such that $||\boldsymbol{y}_m(\mu)|| = \tau$, unless $||\boldsymbol{y}_m(0)|| \leq \tau$, in which case $y_m(0) = B_m^{-1} \boldsymbol{z}_m$ is the solution. Furthermore, $\forall \mu \geq 0$, $\boldsymbol{s}(\mu) = V_m \boldsymbol{y}_m(\mu)$ defines a descend direction for $f(\boldsymbol{x}) = \frac{1}{2} F(\boldsymbol{x})^T F(\boldsymbol{x})$ for $\boldsymbol{x}$, as long as $\boldsymbol{z}_m \neq 0$.

The proof is made for Lemma 4.1 in [1].
In the case in which $||\mathbf{y}_m(0)|| > \tau$, we can not determinate μ such that $||\mathbf{y}_m(\mu)|| = \tau$, so we solve (4.7) approximately. For example, there is a dogled strategy [12] that makes a piecewise linear approximation to the curve $\mathbf{y}_m(\mu)$, and takes $\hat{\mathbf{y}}_m$ as the point on this curve for which $||\hat{\mathbf{y}}_m|| = \tau$. We then define $\mathbf{x}_{k+1} = \mathbf{x}_k + \hat{\mathbf{d}}$, where $\hat{\mathbf{d}} = V_m \hat{\mathbf{y}}_m$. If the iterate $\mathbf{x}_{k+1}$ satisfied a condition like this

$$f(\mathbf{x} + \bar{\mathbf{d}}) \leq f(\mathbf{x}) + \alpha \nabla f(\mathbf{x})^T \bar{\mathbf{d}}$$

with $0 < \alpha < 1$, we proceed to the next step, otherwise, a new value of the trust region size τ is chosen, and the procedure is repeated.

Chapter 4

New method for constrained non linear problems

In general we are interested in nonlinear problems in which variables are constrained and, in particular, this is what happens in out test problem. One of the ways to guarantee that the approximation of the solution respects the constraints, is to project the solution into the domain at each Newton's step, landing at the so-called projected Newton method.
In this chapter we introduce a new projected Newton-Krylov method for nonlinear problems, suggested in a recent paper called "Globalization technique for projected Newton-Krylov methods" published by Jinhai Chen and Cornelis Vuik [4]. The purpose is to fix the biggest defect of the already existing projected Newton-Krylov method, that is no guarantee of convergence, keeping all the advantages. Since, as we will see, the projected Newton direction is not always a descent one, while, if multiplied by a suitable scalar, the projected gradient direction it is , the idea is to switch on the second one all the times the fist one is not satisfactory.
It was already known in theory that a sequence of $\{\mathbf{x}_k\}$, computed in this way $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{d}_k$, with α a suitable length step and $\mathbf{d}_k$ a projected Newton direction of $\Theta(\mathbf{x}_k) = \frac{1}{2}||F(\mathbf{x}_k)||^2$, converges to a stationary point of $\Theta(\mathbf{x})$. What the authors of [4] add to the assertion, is that with a mixed

use of projected Newton and projected gradient direction, $\{\mathbf{x}_k\}$ still ends in a stationary point, and more important, if the second direction is used only finitely many times, then the sequences arrives in a zero of $\Theta(\mathbf{x}_k)$, which is what we are interested in.
We report also two examples of the paper, one analytical, that shows a situation in which projected gradient direction is better that the projected Newton one, and the other numerical, supposed to show the performance of the method. It was not easy to replay this one by ourselves, because of lack of information, and in the end we realized that results shown in the paper were not compatible with the features of the concerned method, so they were wrong. However at least we verified that, for that specific case, the use of projected gradient direction was essential, even if it was slow.
All this analysis make us think that the strong point of this method is robustness rather then velocity.

1 Projected Newton-Krylov method

The framework that we are going to consider is discretized system of nonlinear reaction diffusion equations in which the solutions has to be constrained in a certain interval.
A typical application was studied by van Veldhuizen, Vuik and Kleijn in [15]. They implemented the mathematical model of chemical vapor deposition, that is divided in three parts: Navier-Stokes equations, energy equations and advection-diffusion equations, that model the iterations of reactive species. The last part, discretized implicitly in time in order to guarantee stability, leads to a large-scale system of strongly nonlinear algebraic equations that need to be solved in each time step. Moreover, the variables are species mass fraction, so non-negativity is required for them, and the nonlinear system takes the following form.

$$\begin{cases} F(\mathbf{x}) = 0 \\ \mathbf{x} \geq 0, \end{cases}$$

with $F : \mathbb{R}^n \to \mathbb{R}^n$ continuously differentiable and with around 10^9 unknowns. The authors applied a *projected Newton-Krylov method* to solve the system. What we mean with that name is defined in the Algorithm 1.
In line 3 the forcing term η_k is computed with the techniques shown in section

Algorithm 2: Projected Newton-Krylov

1: $\mathbf{x}_0 \in \mathbb{R}^n_+$, $t \in (0,1)$, $\eta_{max} \in [0,1)$, $m_{max} \in \mathbb{N}$ and $0 < \lambda_{min} < \lambda_{max} < 1$
2: **for** $k = 1, 2, \ldots$ until convergence **do**
3: Choose $\eta_k \in [0, \eta_{max}]$ and find $\mathbf{d}_k$ that satisfy
4: $||F(\mathbf{x}_k) + F'(\mathbf{x}_k)\mathbf{d}_k|| \leq \eta_k ||F(\mathbf{x}_k)||$
5: $m = 0$
6: **while** $m < m_{max}$ **do**
7: Choose $\lambda \in [\lambda_{min}, \lambda_{max}]$
8: **if** $||F(\mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k))|| \leq (1 - t\lambda(1 - \eta_k))||F(\mathbf{x}_k)||$ **then**
9: $\mathbf{x}_{k+1} = \mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k)$, $m = m_{max}$
10: **else**
11: $m = m + 1$
12: **end if**
13: **end while**
14: **end for**

3 and in line 7 λ is chosen with line search methods, for example the ones illustrated in section 4.1. And, of course, as the name suggests, for the inexact Newton-method in line 4, it is used an iterative Newton-Krylov method for finding the solution $\mathbf{d}_k$. The other part of the name comes from the presence of $\mathcal{P}$, that is a orthogonal projection on the domain, in this case $[0, \infty)$. So if we need non-negativity, we will have that the ith entry of $\mathcal{P}(\mathbf{x})$ is defined by

$$\mathcal{P}_i(\mathbf{x}) = \begin{cases} x_i & if \quad x_i \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

This is one of the simplest projection operator. More precisely, this projection assigns to each component of $\mathbf{x}$, that is out of the constrains, the value of

the broken restriction. So, in this case, non-negativity is prevent for all the variables of mass fraction.
An other important fact is that the local convergence of the Newton method is not affected, because of the non-expansiveness of the projection operator, as we will see in *(c)* of Lemma 2.

2 Projected Newton-Krylov method mixed with projected gradient direction

2.1 Idea

The work done in [15] shows that projected Newton-Krylov method is successful, where the classical nonlinear solver packages were not easy to use with such a big scale problem. Nevertheless, as the classical Newton method, this method does not guarantee global convergence; indeed, the search direction found by projected-Newton could not be a descent one, so could not minimize the norm of the underlying function F. Just to be more precise, defined $\Theta(\mathbf{x}) = \frac{1}{2}||F(\mathbf{x})||^2$, then $\mathbf{d}$ is a descent direction for $\Theta(\mathbf{x})$ at point $\mathbf{x}$ if

$$\frac{d\Theta(\mathbf{x}+t\mathbf{d})}{dt}|_{t=0} = \nabla\Theta(\mathbf{x})^T\mathbf{d} < 0. \tag{2.1}$$

Here comes the main idea of the numerical method that we have studied, taken from [4]. Knowing that a *projected gradient direction* on a constrained set, that is $\mathcal{P}(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k$, is *usually* a descent one (Lemma 4), every time that the projected Newton-Krylov method is not able to find a descent direction, the algorithm will use a projected gradient one. The method that comes out still keeps the advantages of the projected Newton-Krylov method, as the capacity of solving extremely large-scale problems, the matrix-free operation, and preconditioning techniques, but it has also global convergence. The next step is to see how it happens.

2.2 Properties of the projector operator

Before presenting in details this method, let us take a step back in a more general setting. Consider the following be a constrained system of nonlinear equations

$$\begin{cases} F(\mathbf{x}) = 0 \\ \mathbf{x} \in \Omega, \end{cases}$$

where Ω is a convex constrained set of $\mathbb{R}^n$, such as $\{\mathbf{x} \in \mathbb{R} \mid \mathbf{l} \leq \mathbf{x} \leq \mathbf{u}\}$, $l_i \in \{\mathbb{R} \cup \{-\infty\}\}$ and $u_i \in \{\mathbb{R} \cup \{\infty\}\}$, $l_i < u_i$ for all $i = 1, ..., n$, and $F : \mathbb{R}^n \to \mathbb{R}^n$ is continuously differentiable on Ω.

Let recall some properties of a projection operator.

Lemma 2. *(see [[3], Lemma 2.1]) Let Π be the projection into Θ.*

(a) If $\boldsymbol{z} \in \Omega$ then $(\Pi(\boldsymbol{x}) - \boldsymbol{x}, \boldsymbol{z} - \Pi(\boldsymbol{x})) \geq 0 \ \ \forall \boldsymbol{x} \in \mathbb{R}^n$.

(b) Π is a monotone operator, that is $(\Pi(\boldsymbol{y}) - \Pi(\boldsymbol{x}), \boldsymbol{y} - \boldsymbol{x}) \geq 0$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. If $\Pi(\boldsymbol{y}) \neq \Pi(\boldsymbol{x})$, then strict inequality holds.

(c) $\Pi(\boldsymbol{x})$ is a non-expansive operator, that is, $||\Pi(\boldsymbol{y}) - \Pi(\boldsymbol{x})|| \leq ||\boldsymbol{y} - \boldsymbol{x}||$ for $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$

The next lemma is given by Gafni and Bertsekas in [[8], Lemma 3] and [[7], Lemma 1.a] and proven by Calamai and Moré in [[3], Lemma 2.2].

Lemma 3. *Let Π be a projection into Ω. Given $\boldsymbol{x} \in \mathbb{R}^n$, then function ψ defined by*

$$\psi(\alpha) = \frac{||\Pi(\boldsymbol{x} + \alpha d) - \boldsymbol{x}||}{\alpha}, \ \alpha > 0, \tag{2.2}$$

is antitone (non-increasing).

With these properties, we arrive to say in the next lemma that the projected Newton-Krylov method, mixed with projected gradient direction, is well-defined, that is the projected gradient direction is descent.

Lemma 4. *Suppose that $\boldsymbol{x}_k$ is not a stationary point of $\Theta(\boldsymbol{x}_k) = \frac{1}{2}||F(\boldsymbol{x}_k)||$ and $\lambda \in (0, 1]$. Then $\mathcal{P}(\boldsymbol{x}_k - \lambda \nabla \Theta(\boldsymbol{x}_k)) - \boldsymbol{x}_k$ is a descent direction for $\Theta(\boldsymbol{x}_k)$.*

Proof. Let $\mathbf{x} = \mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)$ and $\mathbf{z} = \mathbf{x}_k$ in the part (a) of Lemma 2. An immediate consequences of the part (a) of that lemma, says that

$$\begin{aligned}
0 &\leq \Big(\Pi\big(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big) - \big(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big), \mathbf{x}_k - \Pi\big(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big)\Big) \\
&\leq \Big(\big(\Pi(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big) - \mathbf{x}_k, \mathbf{x}_k - \Pi\big(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big)\Big) \\
&\quad + \Big(\lambda\nabla\Theta(\mathbf{x}_k), \mathbf{x}_k - \Pi\big(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big)\Big) \\
&\leq -||\Pi\big(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)\big) - \mathbf{x}_k||^2 - \lambda\nabla\Theta(\mathbf{x}_k)^T\big(\Pi(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k\big).
\end{aligned}$$

So we have for our particular projection $\mathcal{P}$

$$\nabla\Theta(\mathbf{x}_k)^T\big(\mathcal{P}(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k\big) \leq -\frac{1}{\lambda}||\mathcal{P}(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k||^2 < 0. \tag{2.3}$$

This indicates that the projected gradient direction is descent for the norm of $F(\mathbf{x}_k)$. □

2.3 Algorithm

We report the expression of the new algorithm.

Algorithm 3: Projected Newton-Krylov with projected gradient direction

1: $\mathbf{x}_0 \in \Omega$, $t, \sigma \in (0,1)$, $\eta_{max} \in [0,1)$, $m_{max} \in \mathbb{N}$, $0 < \lambda_{min} \leq \lambda_0 \leq \lambda_{max} < 1$ and $FLAG_{NG} = 0$

2: **for** $k = 1, 2, \ldots$ until convergence **do**

3: **if** $FLAG_{NG} = 0$ and if a preconditioned Krylov subspace method finds some $\eta_k \in [0, \eta_{max}]$ and a vector $\mathbf{d}_k$ satisfying

$$||F(\mathbf{x}_k) + F'(\mathbf{x}_k)\mathbf{d}_k|| \leq \eta_k||F(\mathbf{x}_k)|| \tag{2.4}$$

then

$m = 0$, $\lambda = \lambda_0$

while $m < m_{max}$ **do**

Choose $\lambda \in [\lambda_{min}, 1]$

if

$$||F(\mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k))|| \leq (1 - t\lambda(1 - \eta_k))||F(\mathbf{x}_k)|| \tag{2.5}$$

then

$\mathbf{x}_{k+1} = \mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k)$, $m = m_{max}$, $FLAG_{NG} = 0$

else

$m = m + 1$, $FLAG_{NG} = 1$

end if

end while

else

$\mathbf{d}_k = -\nabla\Theta(\mathbf{x}_k)$

$m = 0$, $\lambda = \lambda_0$

while $m < m_{max}$ **do**

Choose $\lambda \in (0, 1]$

if

$$\Theta(\mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k)) \leq \Theta(\mathbf{x}_k) + \sigma \nabla\Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k) - \mathbf{x}_k) \tag{2.6}$$

then

$\mathbf{x}_{k+1} = \mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k)$, $m = m_{max}$, $FLAG_{NG} = 0$

else

$m = m + 1$

end if

end while

end if

end for

According to Lemma 4, the first part of (2.6), that is $\Theta(\mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k)) \leq \Theta(\mathbf{x}_k)$, should be always verified. Besides, the "task" of the second part,

$\sigma \nabla \Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k + \lambda \mathbf{d}_k) - \mathbf{x}_k)$, that is always negative, is to ensure that the new point $\mathbf{x}_{k+1}$ makes the norm of F smaller "enough".

2.4 Convergence results

The authors Chen and Vuik display in [4] an interesting result for the convergence of this new method, that is the content of next theorem.

Theorem 2.1. *Assume that $\{\boldsymbol{x}_k\} \subset \Omega$ is a sequence generated by the feasible projected Newton-Krylov method. Then any accumulation point of $\{\boldsymbol{x}_k\}$ is at least a stationary point of $\Theta(\boldsymbol{x}_k)$. Further, if (2.5) is satisfied by the projected Newton direction for all but finitely many k, then $\boldsymbol{x}^*$ is a zero of $F(\boldsymbol{x})$ on Ω.*

Proof. Let $\mathbf{x}^*$ be an accumulation point of a sequence $\{\mathbf{x}_k\}$ generated by our method. The notation for λ that is used in the paper [4] is $\lambda_0^{m_k}$, that indicates that m_k is the power of λ_0, but to be more general, we can just say that $\lambda_0^{m_k}$ is the λ picked up with an arbitrary technique at the m_kth attempt in the line search loop for kth nonlinear iteration. In addition, suppose that the first λ that we try for each nonlinear iteration k is $\lambda_0^0 = 1$.
There are two cases.
First, suppose that the projected Newton direction is used, so (2.5) holds, for finitely many iterations. It follows immediately that $F(\mathbf{x}^*) = 0$ because of the local convergence properties of the projected Newton method, that are the same of the classic one. So $\mathbf{x}^*$ is a stationary point.
Second, suppose that the projected gradient direction is used ((2.6) holds) for all but finitely many iterations. It follows from (2.6) and from Lemma 2, that $\{\Theta(\mathbf{x}_k)\}$ is monotonically decreasing (unless the method terminates at a stationary point at any finite step) and is bounded below by 0. Hence, it converges and

$$\lim_{k\to\infty} (\Theta(\mathbf{x}_{k+1}) - \Theta(\mathbf{x}_k)) = 0.$$

But then, using (2.6),we see that

$$\lim_{k\to\infty} \nabla \Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k} \nabla \Theta(\mathbf{x}_k)) - \mathbf{x}_k) = 0, \tag{2.7}$$

where $\lambda_0^{m_k}$ can be seen as the lambda chosen at step k.
Let $\{\mathbf{x}_k, k \in K\}$ be a subsequence converging to $\mathbf{x}^*$. We have two cases for (2.7). *Case 1*: Assume

$$\liminf_{k(\in K)\to\infty} \lambda_0^{m_k} > 0.$$

By (2.7) and (2.3), it follows that for some infinite subset $K' \subseteq K$,

$$\lim_{k(\in K)\to\infty} -||\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k||^2 = 0.$$

Hence, $\mathbf{x}^*$ is a stationary point of $\Theta(\mathbf{x})$.
Case 2: Assume that there is a subsequence $\{\mathbf{x}_k\}_{k\in J}, J \subseteq K$ with

$$\lim_{k(\in J)\to\infty} \lambda_0^{m_k} = 0.$$

So, for sufficiently large $k(\in J)$, $\mathbf{x}_k$ is not a stationary point. Otherwise $\lambda_0^{m_k}$ should be different from 0 because of (2.6); in fact, if we have a stationary point, the first λ that we try should verify the inequality, that is $\lambda_0^{m_k} = \lambda_0^0 = 1$.
Therefore, for sufficiently large $k(\in J)$, it holds that

$$||\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k|| > 0, \tag{2.8}$$

because $\mathbf{x}_k$ is not a stationary point for all λ taken up to now, therefore also the previous one. In addition, it follows from (2.6) that

$$\Theta(\mathcal{P}(\mathbf{x}_k + \lambda_0^{m_k-1}\mathbf{d}_k)) - \Theta(\mathbf{x}_k) > \sigma\nabla\Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k + \lambda_0^{m_k-1}\mathbf{d}_k) - \mathbf{x}_k).$$

Moreover, by the mean value theorem, we know

$$\begin{aligned}
&\Theta(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))) - \Theta(\mathbf{x}_k)\\
&\quad=\nabla\Theta(\boldsymbol{\xi}_k)^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k)\\
&\quad=(\nabla\Theta(\boldsymbol{\xi}_k) - \nabla\Theta(\mathbf{x}_k))^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k)\\
&\qquad+ \nabla\Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k)\\
&\quad>\sigma\nabla\Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k),
\end{aligned}$$

where $\boldsymbol{\xi}_k$ is a point in the line segment between $\mathbf{x}_k$ and $\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))$, so $\boldsymbol{\xi} = \tau\mathbf{x}_k + (1-\tau)\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))$ for some $\tau \in (0,1)$. Consequently,

$$\begin{aligned}(\nabla\Theta(\boldsymbol{\xi}_k) - \nabla\Theta(\mathbf{x}_k))^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k) \\ > (1-\sigma)\nabla\Theta(\mathbf{x}_k)^T(\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))).\end{aligned}$$

Further,

$$\begin{aligned}&\nabla\Theta(\mathbf{x}_k)^T(\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))) \\ &\quad < \frac{1}{1-\sigma}(\nabla\Theta(\boldsymbol{\xi}_k) - \nabla\Theta(\mathbf{x}_k))^T(\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k) \\ &\quad < \frac{1}{1-\sigma}||\nabla\Theta(\boldsymbol{\xi}_k) - \nabla\Theta(\mathbf{x}_k)||||\mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k||.\end{aligned} \tag{2.9}$$

From Lemma 3, we know that $\frac{||\mathbf{x}_k - \Pi(\mathbf{x}_k - \lambda\nabla\Theta(\mathbf{x}_k))||}{\lambda}$ is monotonically nonincreasing with respect to λ. From (2.3), we know that

$$\begin{aligned}&\nabla\Theta(\mathbf{x}_k)^T(\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))) \\ &\quad \geq \frac{||\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))||^2}{\lambda_0^{m_k-1}} \\ &\quad \geq \frac{||\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0\nabla\Theta(\mathbf{x}_k))||}{\lambda_0}||\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0^{m_k-1}\nabla\Theta(\mathbf{x}_k))||,\end{aligned}$$

with the assumption that $\lambda_0 \equiv \lambda_0^1 \geq \lambda_0^{m_k-1}$.
This combined with (2.8) and (2.9), implies

$$\frac{||\mathbf{x}_k - \mathcal{P}(\mathbf{x}_k - \lambda_0\nabla\Theta(\mathbf{x}_k))||}{\lambda_0} < \frac{1}{1-\sigma}||\nabla\Theta(\boldsymbol{\xi}_k) - \nabla\Theta(\mathbf{x}_k)||.$$

Passing to the limit as $k(\in J) \to \infty$, we obtain

$$\frac{||\mathcal{P}(\mathbf{x}_* - \lambda_0\nabla\Theta(\mathbf{x}_*)) - \mathbf{x}_*||}{\lambda_0} = 0,$$

which implies $\mathbf{x}^*$ is a stationary point. Therefore, in either case, we establish the assertion. □

The next result is about local convergence, but first we need to recall some concepts.

If $F(\mathbf{x})$ is continuously differentiable, then $F(\mathbf{x})$ is locally Lipshitz continuous at $\mathbf{x}$, that is, there exists a constant $L_{\mathbf{x}}$ such that for all $\mathbf{y}$ sufficiently close to $\mathbf{x}$,

$$||F(\mathbf{y}) - F(\mathbf{x}))|| \leq L_{\mathbf{x}}||\mathbf{y} - \mathbf{x}||.$$

Further, if $F'(\mathbf{x})$ is nonsingular at $\mathbf{x}^*$, so there exists a constant $C_{\mathbf{x}^*}$ such that $||F'(\mathbf{x}^*)^{-1}|| \leq C_{\mathbf{x}^*}$, then there is a costant C such that $F'(\mathbf{y})^{-1}$ exists, and

$$||F'(\mathbf{y})^{-1}|| \leq C$$

for all $\mathbf{y}$ sufficiently close to $\mathbf{x}^*$.

Theorem 2.2. *Assume that $\boldsymbol{x}^* \in \Omega$ is a limit point of $\{\boldsymbol{x}_k\}$ generated by the feasible projected Newton-Krylov method. Assume also that $F(\boldsymbol{x}^*) = 0$ and $F'(\boldsymbol{x}^*)$ is nonsingular. Then the wole sequence $\{\boldsymbol{x}_k\}$ converges to $\boldsymbol{x}^*$. Furthermore, for large enough k,*

(a) if η_{max} and t are chosen by

$$\begin{cases} \eta_{max} \in (0,1), \;\; t \in [C^2L, 1), \quad if \;\; C^2L < 1, \\ \eta_{max} \in (0, \frac{1-t}{C^2L-t}), \;\; t \in (0,1), \quad if \;\; C^2L \geq 1, \end{cases}$$

where L is the Lipschitz constant of F at $\boldsymbol{x}^*$ and C is an upper bound of inverse of $F(\boldsymbol{x})$ defined in a neighborhood of $\boldsymbol{x}^*$, then the projected Newton direction is eventually accepted with $\lambda = 1$, that is, no projection gradient is carried out.

(b) if

$$\begin{cases} \eta_{max} \in (0, \min\{\frac{1}{CL}, 1\}), \;\; t \in [C^2L, 1), \quad if \;\; C^2L < 1, \\ \eta_{max} \in (0, \min\{\frac{1}{CL}, \frac{1-t}{C^2L-t}\}), \;\; t \in (0,1), \quad if \; C^2L \geq 1, \end{cases}$$

then the convergence rate is Q-linear;

(c) if $\eta_k \to 0$, the convergence rate is Q-superlinear.

2.5 Previuos thoery of convergence for projected gradient direction

Properties of projected gradient direction have been studied since 1960 and, in particular, we propose the handling of paper [3]. As we already know, gradient projection algorithm is defined by:

$$\mathbf{x}_k = P(\mathbf{x}_k - \alpha_k \nabla\Theta(\mathbf{x}_k)). \tag{2.10}$$

In the convergence analysis of [3], there is a theorem (Theorem 2.4) that states that if $\Theta : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable on Ω, $\{\mathbf{x}_k\}$ be the sequence generated by (2.10), with a suitable choice of α_k and if some subsequence $\{x_k, k \in K\}$ is bounded, then

$$\lim_{k\in K, k\to\infty} \frac{||\mathbf{x}_{k+1} - \mathbf{x}_k||}{\alpha_k} = 0$$

and any limit point of (2.10) is a stationary point of problem $\Theta(\mathbf{x})$. This result means it was already known that if we use only projected gradient direction, the sequence of $\mathbf{x}_k$ is going to converge in a stationary point. Theorem 2.1 adds that the mix of projected Newton directions and projected gradient directions ends still in a stationary point, moreover, if the second ones are used only finitely many times, it reaches a point in which $\Theta(\mathbf{x}) = 0$, that, actually, is our concern.
In general the projected gradient of $\Theta(\mathbf{x})$ is defined by

$$\nabla_\Omega\Theta(\mathbf{x}) \equiv argmin\{||v + \nabla\Theta(\mathbf{x})|| : v \in T(\mathbf{x})\}, \tag{2.11}$$

where $T(\mathbf{x})$ is the *tangent cone*, closure of the cone of feasible directions. In our case $\nabla_\Omega\Theta(\mathbf{x}_k) = P(\mathbf{x}_k - \nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k$. It is shown that this operator has the following properties.

Lemma 5. *Let $\nabla_\Omega\Theta(\boldsymbol{x})$ be the projected gradient of Θ at $\boldsymbol{x} \in \Omega$.*

(a) $-(\nabla\Theta(\boldsymbol{x}), \nabla_\Omega\Theta(\boldsymbol{x})) = ||\nabla_\Omega\Theta(\boldsymbol{x})||^2$.

(b) $\min\{(\nabla\Theta(\boldsymbol{x}), v) : v \in T(\boldsymbol{x}), ||v|| \leq 1\} = -||\nabla_\Omega\Theta(\boldsymbol{x})||$.

(c) The point $\boldsymbol{x} \in \Omega$ is a stationary point of $\Theta(\boldsymbol{x})$ if and only if $\nabla_\Omega \Theta(\boldsymbol{x}) = 0$.

Point (b) is important because it is displays that $\nabla_\Omega \Theta(\mathbf{x})$ is a steepest descent direction for Θ, so we can say that our projected gradient direction $P(\mathbf{x}_k - \nabla\Theta(\mathbf{x}_k)) - \mathbf{x}_k$ is still the steepest one in Ω.

2.6 Examples

Analytical example

We report one simple example that shows how projected gradient direction can "save" situations in which the projected Newton direction cannot be descent. Consider the following constrained problem for $\mathbf{x} = (x_1, x_2)^T \in \Omega = (-\infty, 1] \times (-\infty, 1]$:

$$\begin{cases} x_1^2 - x_2 - 2 = 0, \\ x_1 - x_2 = 0, \end{cases}$$

with $\mathbf{x}^* = (-1, -1)^T$ unique solution in Ω. We have :

$$F'(\mathbf{x}) = \begin{pmatrix} 2x_1 & -1 \\ 1 & -1 \end{pmatrix} \qquad ,$$
$$\nabla\Theta(\mathbf{x}) = (2x_1(x_1^2 - x_2 - 2) + x_1 - x_2, \quad -(x_1^2 - x_2 - 2) - (x_1 - x_2))^T.$$

If we take $\mathbf{x}_0 = (1, \frac{1}{2})^T$, then it follows that the Newton direction is $\mathbf{d}_0 = (2, \frac{5}{2})^T$ with $\eta_0 = 0$ in Algorithm 3. As we know from theory, this direction, if calculated exactly, is a descent one; indeed:

$$\nabla\Theta(\mathbf{x}_0)^T \mathbf{d}_0 = \left(-\frac{5}{2}, 1\right) \cdot \left(2, \frac{5}{2}\right)^T = -\frac{5}{2} < 0.$$

The problem comes when we project it because we find

$$\begin{aligned} P(\mathbf{x}_0 + \lambda \mathbf{d}_0) - \mathbf{x}_0 &= \left(\min\{1 + 2\lambda, 1\}, \min\{1, \tfrac{1}{2} + \tfrac{5\lambda}{2}\}\right)^T - \left(1, \tfrac{1}{2}\right)^T \\ &= \left(0, \min\{\tfrac{1}{2}, \tfrac{5\lambda}{2}\}\right), \end{aligned}$$

and therefore

$$\begin{aligned}\nabla\Theta(\mathbf{x}_0)^T(P(\mathbf{x}_0+\lambda\mathbf{d}_0)-\mathbf{x}_0) &= \left(-\tfrac{5}{2},1\right)\cdot\left(0,\min\{\tfrac{1}{2},\tfrac{5\lambda}{2}\}\right)^T \\ &= \min\{\tfrac{1}{2},\tfrac{5\lambda}{2}\}\geq 0,\end{aligned}$$

for any $\lambda\in(0,1]$. We conclude that in this case the projected Newton direction can not be descent, but, as we expect from Lemma 4, the projected gradient direction has to be a good one; indeed:

$$\begin{aligned}P(\mathbf{x}_0-\lambda\nabla\Theta(\mathbf{x}_0))-\mathbf{x}_0 &= \left(\min\{1+\tfrac{5}{2}\lambda,1\},\min\{1,\tfrac{1}{2}-\lambda\}\right)^T-\left(1,\tfrac{1}{2}\right)^T \\ &= (0,-\lambda)^T,\end{aligned}$$

and

$$\nabla\Theta(\mathbf{x}_0)^T(P(\mathbf{x}_0-\lambda\nabla\Theta(\mathbf{x}_0))-\mathbf{x}_0)=\left(-\frac{5}{2},1\right)\cdot(0,-\lambda)^T=-\lambda<0.$$

The geometrical scenario is illustrated in Figure 2.1, where we see that the projected Newton direction (PND) cannot be descent, because it lies on the semi-plane in which the function is supposed to grow, and that the projected gradient (PGD) is descent.

Numerical example

In addiction to the analytical example that we have just shown, there is also a numerical one in [4] (Example 8), whose aim is to show the performances of the new projected Newton-Krylov method. It is about a system of nonlinear equations for $\mathbf{x}=(x_1,\ldots,x_n)^T$ with $\Omega=\{\mathbf{x}\in\mathbb{R}^n|x_1\in[0.8,2],x_i\in[0.5,2],2\leq i\leq n\}$. The system is

$$\begin{cases}x_1^2-1=0,\\ x_1-x_2^3=0,\\ x_2-x_3^3=0,\\ \quad\vdots\\ x_{n-2}-x_{n-1}^2=0,\\ x_{n-1}-x_n=0\end{cases}$$

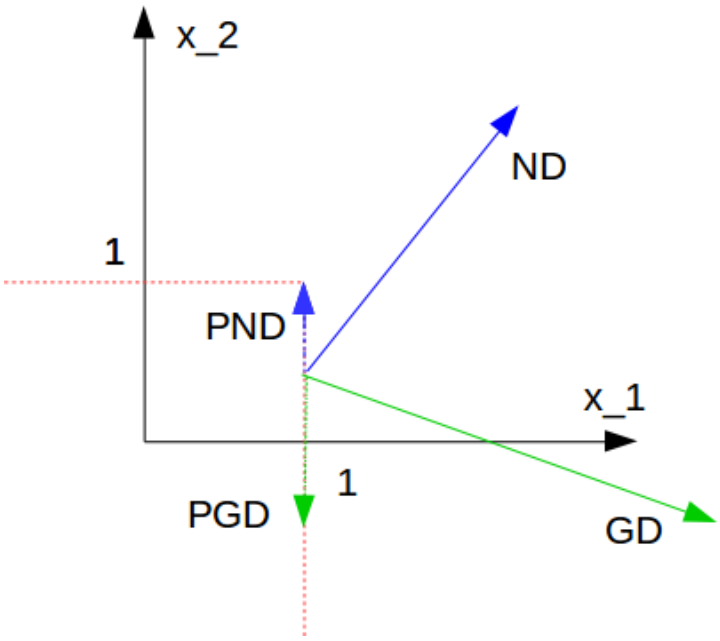


Figure 2.1: Blue arrows are Newton directions (projected and not), while green ones are gradient directions (projected and not). We see how the projected Newton direction (PND) is increasing for the norm of F since it lies in the semi-plane in which the function is supposed to grow, that is the semi-plane that has the opposite of GD. Instead, the projection of gradient direction (PGD) is in the decreasing semi-plane.

with solution $\mathbf{x}^* = (1, \ldots, 1)^T$ in Ω.
As a first approach, we tried to implement this problem by our self with $n = 100$ in order to compare our results with the ones of the paper. The information that we had about settings of Algorithm 3, was the following:

(a) initial guess for the nonlinear iterations $\mathbf{x}_0(1:20) = 0.9$ and $\mathbf{x}_0(21:100) = 0.5$, and zero vector for the linear ones;

(b) termination tolerance rule for nonlinear iteration: $||F(\mathbf{x}_k)|| \leq 10^{-12}$;

(c) choice of linear search parameter $\lambda = \lambda_0^m$, with $\lambda_0 = 0.5$ for porjected Newton direction and $\lambda_0 = 0.8$ for projected gradient one.

(d) $\sigma = t = 10^{-4}$, $m_{max} = 20$, $\eta_{max} = 0.9$;

(e) linear solver GMRES.

For the choice of forcing term η_k the authors just say that they used one of the options in [6], the ones that we illustrated in the previous chapter. The problem for us was that we did not know exactly which one was the choice used and which was the value of the parameters. Using all information we had and the values of residual norm $||F(\mathbf{x})||$ that we should have obtained (table in Figure 2.2), we applied a reverse iterative technique to find out the forcing terms. In practice, we interpolated the tolerance needed in the linear solver (so the forcing term) to arrive to the residual norms of Figure 2.2 and, after few steps, we found that they used Choice 2 (3.4) with $\alpha = 2$, $\gamma = 0.9$ and $\eta_0 \simeq 0.765518$.

First of all, we have to point out that in Figure 2.2, each value of $||F(\mathbf{x})||$ is actually the input value of that iteration, so, for example, 3.4873 corresponds to $||F(\mathbf{x}_0)||$, and the value in the second iteration is the residual norm that comes from the first iteration. The same is true for values of λ , so the columns of λ and $||F(\mathbf{x})||$ are shifted by one iteration respect to the other two columns.

After all this procedure, we realized that the values of λ in iterate 5 and 6 are reported wrongly, indeed in 5, λ is equal actually to 0.5^2 and not to 0.8^2 because it refers to the previous step that uses PN, and not PG. Indeed in 6, λ is equal to 0.8 and not to 0.5 for the same reason. This correction needs to be done for iterates 9, 10, 13 and 14 as well. We have reported our results in Figure 2.3 in the same format as in Figure 2.4.

Another thing that we noticed, more important, was that, in order to have the same results as in the paper's Table, we had to force λ to be equal to 0.8 in each time PG was used, even if (2.6) was not verified. When we did the test as Algorithm 3 indicates, at iteration 5, $\lambda = 0.8$ was not enough, so the algorithm selected $\lambda = 0.8^4$ to satisfy (2.6). Therefore, all the convergence behavior, that follows step 5, changes. Figure 2.4 and 2.5, shown their and our trend of the residual norm.

Iterates	λ	$\|F(x)\|$	Search direction
1	1	3.4873	PN
2	0.2500	3.4840	PN
3	0.1250	3.4813	PN
4	0.1250	3.4796	PN
5	0.6400	3.4779	PG
6	0.5000	3.8144	PN
7	1	3.6715	PN
8	1	3.6174	PN
9	0.6400	3.6087	PG
10	0.5000	4.1736	PN
11	1	3.7805	PN
12	1	3.7395	PN
13	0.6400	3.7388	PG
14	0.5000	4.9133	PN
15	1	3.8563	PN
16	0.0039	3.5496	PN
17	0.1250	2.3928	PN
18	1	0.5615	PN
19	1	0.1172	PN
20	1	0.0037	PN
21	1	2.6134e-06	PN
22	1	1.1652e-12	PN
23	1	3.8432e-13	PN

Figure 2.2: Table of results of Example 8 in [4], PN indicates that it was used projected Newton direction and PG projected gradient direction. The values of columns of λ and $||F(\mathbf{x})||$ are shifted by one respect to the others; indeed, for example, the norm of F, that was achieved in iteration 1, and the λ, that was used, are the ones reported in the second row.

Actually, there was no need to implement all by our self to see that there is something wrong. In fact, if we notice the trend of the residual norm in Figure 2.2 and 2.4, we realize that is not non-increasing. That should not happen when (2.6) is verified, since σ has to be positive and $\nabla\Theta(\mathbf{x}_k)^T(\mathcal{P}(\mathbf{x}_k+\lambda\mathbf{d}_k)-\mathbf{x}_k)$ has to be negative because the projected gradient direction is a descent one (Lemma 4). So the residual norm has to be strictly decreasing.

We have discovered that in our implementation much more iterations were required to reach convergence and for most of them it is used projected gradient direction, so the decrease of residual norm is very slow. Indeed the

Iterates	lambda	||F||	eta	flag
00001	1	3.4873	0.76552	PN
00002	0.25	3.484	0.76552	PN
00003	0.125	3.4813	0.89833	PN
00004	0.125	3.4796	0.8986	PN
00005	0.25	3.4779	0.8991	PG
00006	0.8	3.8144	0.89912	PN
00007	1	3.6715	0.9	PN
00008	1	3.6174	0.83384	PN
00009	0.25	3.6087	0.87367	PG
00010	0.8	4.1736	0.89567	PN
00011	1	3.7805	0.9	PN
00012	1	3.7395	0.73843	PN
00013	0.25	3.7388	0.8806	PG
00014	0.8	4.9133	0.89965	PN
00015	1	3.8563	0.9	PN
00016	0.0039062	3.5496	0.729	PN
00017	0.125	2.374	0.76255	PN
00018	1	0.55595	0.52333	PN
00019	1	0.11578	0.24649	PN
00020	1	0.0036353	0.039035	PN
00021	1	2.5603e-06	0.00088721	PN
00022	1	1.0132e-12	4.4643e-07	PN

Figure 2.3: Table of our results for Example 8 in [4]. We managed to generate this results, that are equal to the ones in Figure 2.2, at least in the first 16 iterations. With a revers iterative technique we found missed information, that is the choice for η_k. But, in order to obtain the same values as in Figure 2.2, we had also to impose $\lambda = 0.8$ in each PG step, even if the inequality (2.6) was not satisfied.

use of PG cannot be considered as a massive one because it is too slow, it should intervene in isolated steps when PN is not able to find a acceptable direction.

In any case, we tried to implement the algorithm without PG and we verified that it is not going to converge because the PN forces $\mathbf{x}$ to arrive to the other solution $\mathbf{x}^* = (-1, \ldots, -1)^T$, that is out of Ω, so $\mathbf{x} + \lambda\mathbf{d}$ is always projected on the constraints, consequently the algorithm remains stuck. Figure 2.5 predicts a quite slow method, but at least robust, because it manage to converge to the domain's solution.

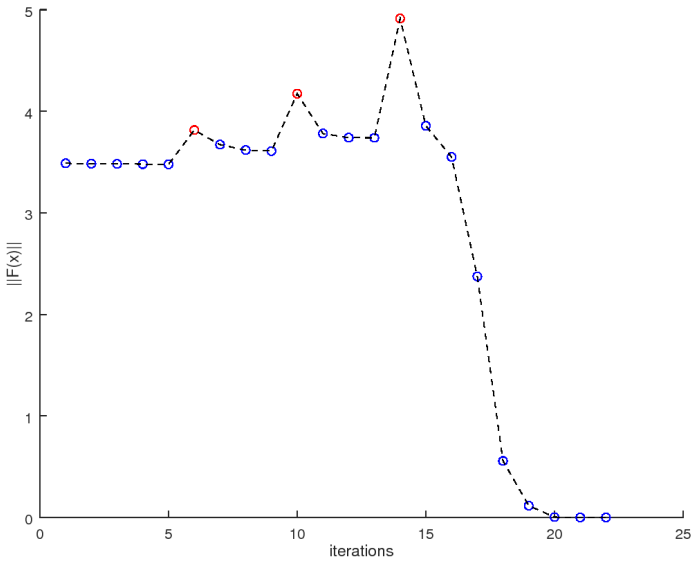


Figure 2.4: Residual norms according to paper's result, blu point are PN and red ones are PG. We see that they does not respect the main characteristic of the analyzed method, which is that the norm of F has not to increase. This happens because inequality (2.6) is not satisfied.

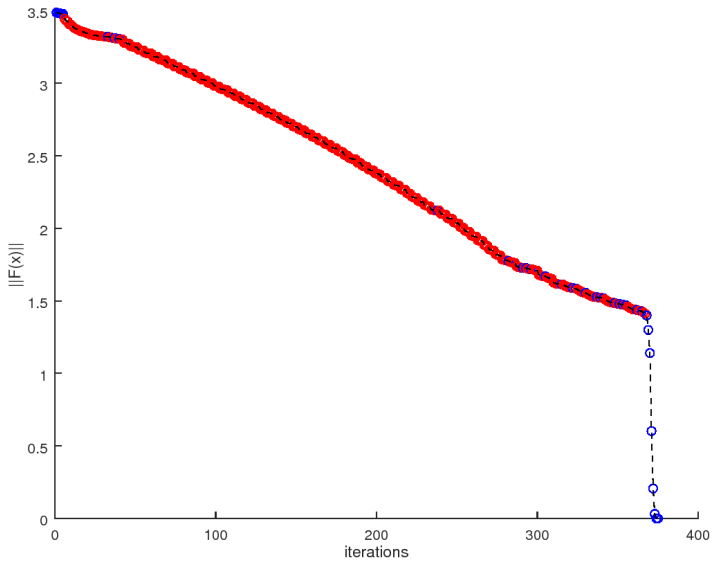


Figure 2.5: Residual norms that we obtained implementing Algorithm 3 in the right way. In this case the number of iterations is much bigger than the one promised in the paper and a lot of PG are used, reason for which the convergence is slow. However, without the use of PG, it does not converge, because at each step Newton direction tries to push in the direction of the solution out of Ω, that is $\mathbf{x} = (-1, -1, \ldots, -1)^T$, but the result is always projected on the constraints. That brief analysis predicts a quite slow method, but at least robust.

Chapter 5

Results

Up to now we have chosen and illustrated the main ingredients for the resolution of nonlinear coupled systems of partial differential equations and now we can recall the propose, that we declared in Chapter 1, and complete it. Therefore the main passages were:

- time discretization: time adaptivity that at each iteration calculates the longest time step that permits to have convergence of order 1;
- space discretization: quadtrees mesh and variables ordered by nodes to facilitate parallel solving based on the mesh;
- nonlinear method: projected Newton-Krylov method mixed with gradient direction.

Now it is time to put all them into practice and to test how they work together on the test problem of Chapter 1. First of all we will try to make some one-dimensional simulations in order to see if our results are coherent with theory and with the previous studies, and then we will try do stimulate differently the problem in order to analyze more deeply our strategy.
Let fist specify the parameter's values we have used in Algorithm 3 of the previous Chapter for all the simulations, that are:

- $t = 10^{-4}$, $\sigma = 10^{-4}$, $\eta_{max} = 0.9$, $mmax = 10$, $\lambda_0 = 1$;

- Choice 2 for the calculation of η_k with $\alpha = 2$ and $\gamma = 1$;
- Krylov method: GMRES;
- $\lambda = \lambda_0^m$ at each iteration m of the line search;
- norm of increments as convergence condition of the nonlinear iterations with tolerance equal to 10^{-6}.
- tolerance for the time adaptivity equal to 10^{-3}.

Last clarification that we have to do is that the Jacobian of our test problem (2.1) is an approximation; indeed if we consider

$$\begin{cases} F_1(m,n) = 0 \\ F_2(m,n) = 0 \\ m, n \geq 0, \end{cases}$$

the exact Jacobian has this expression:

$$DF(m,n)[\delta\mathbf{m}, \delta\mathbf{n}] = \begin{cases} DF_1(m,n)[\delta\mathbf{m}, \delta\mathbf{n}] \\ DF_2(m,n)[\delta\mathbf{m}, \delta\mathbf{n}], \end{cases}$$

with

$$\begin{aligned} DF_1(m,n)[\delta\mathbf{m}, \delta\mathbf{n}] = &- \mu\nabla(mK_\gamma\nabla(\gamma(m+n)^{\gamma-1}(\delta\mathbf{m}+\delta\mathbf{n}))) - \mu\nabla(K_\gamma\nabla((m+n)^\gamma)\delta\mathbf{m}) \\ &- G(p)\delta\mathbf{m} - G'(p)\frac{\partial p}{\partial m}m\delta\mathbf{m} - G'(p)\frac{\partial p}{\partial n}m\delta\mathbf{n}; \\ DF_2(m,n)[\delta\mathbf{m}, \delta\mathbf{n}] = &- \nu\nabla(mK_\gamma\nabla(\gamma(m+n)^{\gamma-1}(\delta\mathbf{m}+\delta\mathbf{n}))) - \nu\nabla(K_\gamma\nabla((m+n)^\gamma)\delta\mathbf{n}). \end{aligned}$$

What we actually do is to neglect $\nabla(m+n)$, which means that the part of $-\mu\nabla(K_\gamma\nabla((m+n)^\gamma)\delta\mathbf{m})$ is thrown away and also a part of the first term. Since the characteristics of gradient direction are based on the veracity of the Jacobian, we tried to make tests also with the complete Jacobian, but we saw that almost nothing changes.

1 One-dimensional wave

For the one dimension case, introduced in Section 2.2 of Chapter 1, we evaluated the solution in $t = 0$ and we set it as initial condition to see if theory can confirm our results. Using the same initial supports for m and n, with $P_M = 25$, $\gamma = 30$, $\mu = 0.5$ and $\nu = 1$, and a mesh of 600 nodes for a interval $[-2, 10]$, a traveling wave of dividing cancerous cells has occurred. Figure 1.1 illustrated the graphics of m, n and $\frac{p}{P_M}$ in the times $t = 0$ (sx) and $t = 0.4$ (dx). What we can observe is that wavefront's average velocity is circa 10, since in a time interval of 0.4, the wave arrives in $x = 4$. This is quite near to the expected value, that from theory, (2.8) in Chapter 1 , is equal to 10.35.

This is also confirmed in the plot of pressure p and absolute value of its

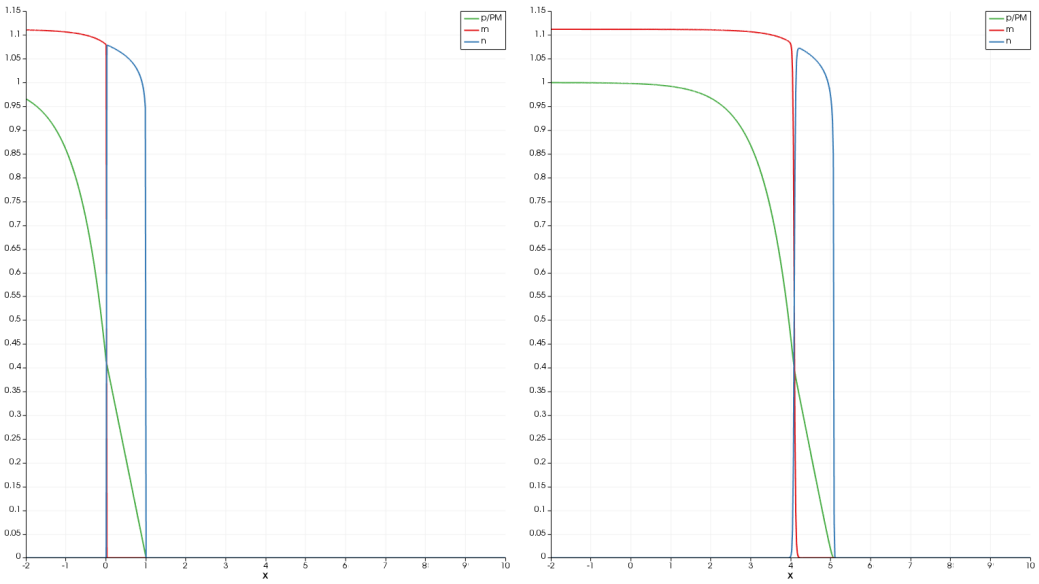


Figure 1.1: The graphics illustrate m, n and $\frac{p}{P_M}$ in $t = 0$ (sx) and $t = 0.4$ (dx), for a case $\mu < \nu$. Separation between the two types of cells persists during the wave motion and the average velocity is circa 10, which is quit near to the theoretical value of σ (10.35).

derivative (Figure 1.2); indeed on the border of the two supports, where p' is discontinuous, we can verify from the values of the graphic that conditions

$p'(0^-) = \frac{\nu}{\mu} p'(0^+) = -\frac{\sigma}{\mu}$ are satisfied.

If we swap the values of μ and ν, we see in Figure 1.3 that healthy cells

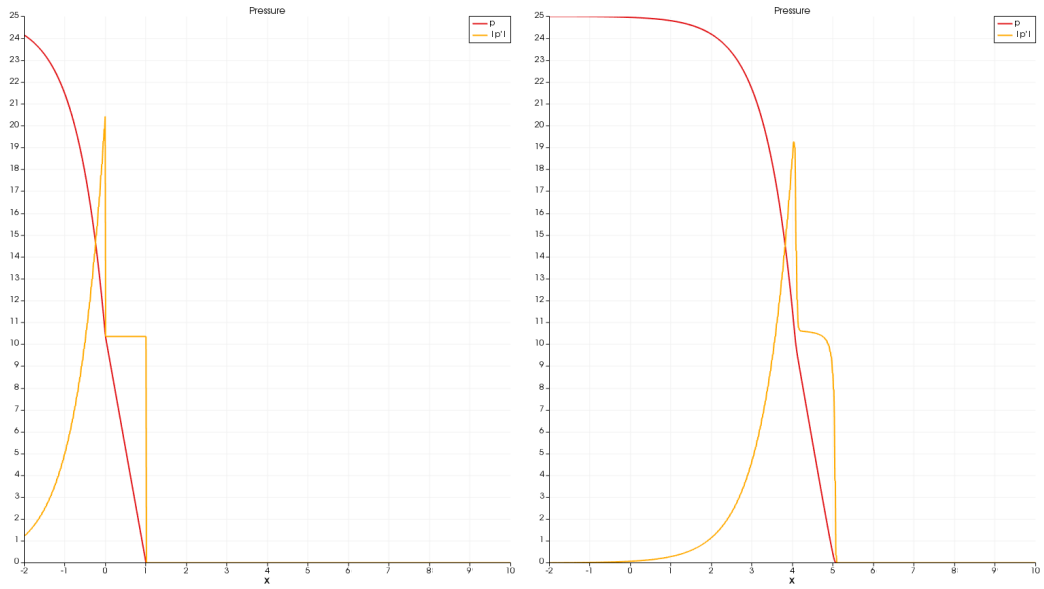


Figure 1.2: The graphics illustrate p and $|p'|$ in $t = 0$ (sx) and $t = 0.4$ (dx), for a case $\mu < \nu$. On the discontinuity of the pressure derivatives, it is seems that conditions $p'(0^-) = \frac{\nu}{\mu} p'(0^+) = -\frac{\sigma}{\mu}$ are verified.

are not able to contrast cancerous one on the wavefront, so they just remain behind; indeed this is the case in which the mobility of dividing cells is bigger than the one of non-dividing.

2 Bi-dimensional wave

Setting all the variables as they are in Section 2.3 of Chapter 1, we tried to simulate the case of a spherical wave with $\mu = 1$ and $\nu = 2$. What we obtain, with a mesh 128×128 is shown in Figure 2.1.

We also tried to simulate the case of a straight wave front, instead of a spherical one with initial conditions :

$$m(x, y, t = 0) := a_m e^{-b_m(x^2)} \quad \text{and} \quad n(x, y, t = 0) := a_n e^{-b_n(x^2)}. \tag{2.1}$$

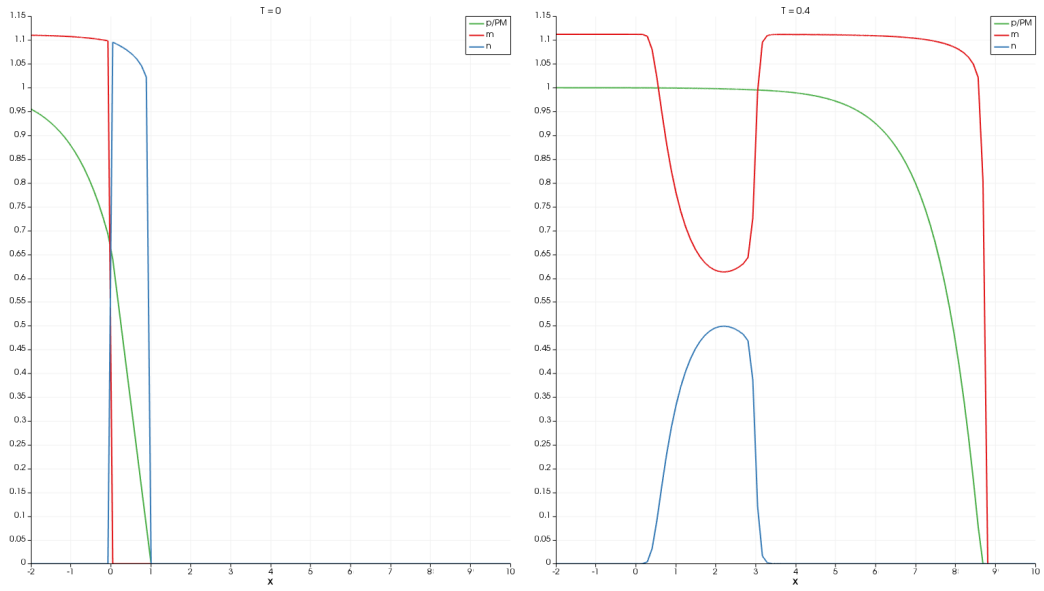


Figure 1.3: The graphics illustrate m, n and $\frac{p}{P_M}$ in $t = 0$ (sx) and $t = 0.4$ (dx), for a case $\mu > \nu$. The fact that cancerous cells have greater mobility does not permit to healthy ones to stay on the wave's interface and they are left behind. That allows the cancer expansion to have greater velocity.

We wanted to see if, after a long time of propagation, the wavefront could have a characteristic perturbation, but actually it did't happen. Therefore we implemented a initial condition in which the wavefront was already perturbed, and we waited to see what happens in a case of $\mu < \nu$ and vice versa. The initial condition is the one of Figure 2.2. What we obtain in the first scenario ($\mu < \nu$), after a time's interval of 1, is a straight separation between m and n, especially on the front of the two ledges in the dividing cells' propagation. Figure 2.3 shows the result.

Instead what happens with the unstable case, that is $\mu > \nu$, is anticipate in the one dimensional case; indeed here the cancerous cells are able to "pass over" the healthy ones, leaving them behind and propagating undisturbed. When we look at the motion, we notice also that, left the concentration of n behind, the wavefront of m becomes straight again (see Figure 2.4)

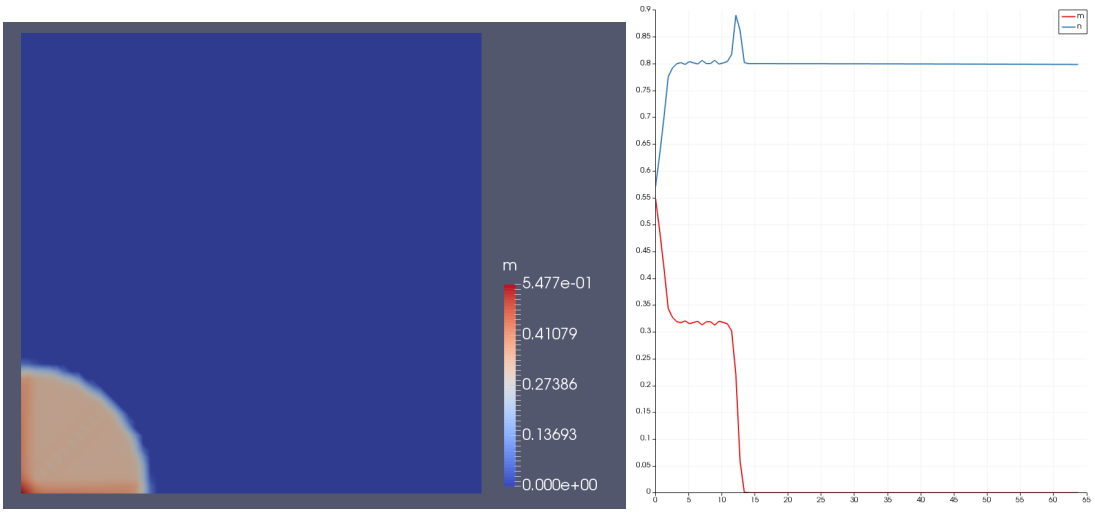


Figure 2.1: Plot in two dimensions of the spherical tumorous wave and its diagonal section at time $T = 1$.

3 Parallel scaling

Since we also did a substantial work to parallelize the resolution of such problems, we run the same test with different number of processors, in order to see how the computing time is scaled. We did tests with a two-dimensional square mesh, 256×256, with the same settings as in Section 2.3 of Chapter 1, but with a straight wave front, instead of a spherical one. Therefore the initial conditions are the one in (2.1).
First we have run tests with 1, 2 and 4 processors for 3 hours each, and then we compared them taking the last time of the problem computed and seeing where the wave of dividing cells has been arrived. Figures 3.1a, 3.1b and 3.1c show the result, where T is the last time reached.

We also ran the same test problem for more then thousand iterations with different number of processors, from 1 to 8, and then we compared the average computational time. To be more precise, we select tree main passages of the resolution of each nonlinear step:

- *Assemble lhs and rhs*: at the beginning of each nonlinear step, the Jacobian matrix (lhs) and the rhs vector are assembled from the initial

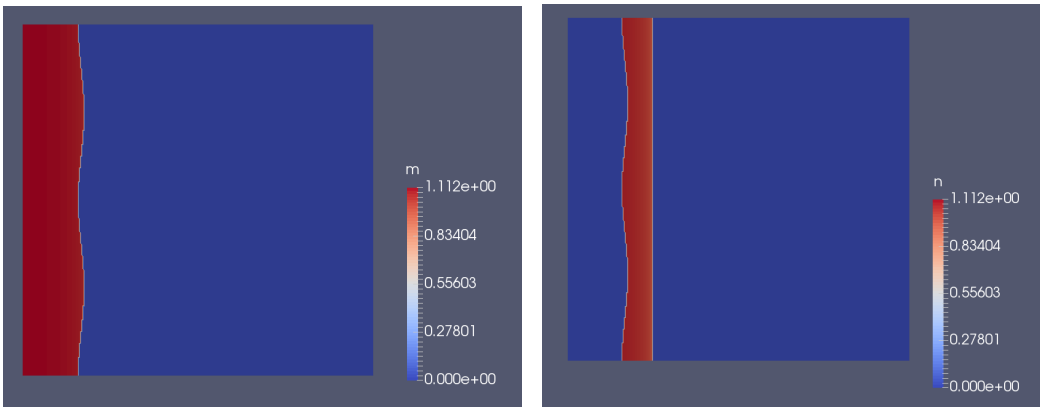


Figure 2.2: Initial condition with a perturbated front wave, m (sx) and n (dx).

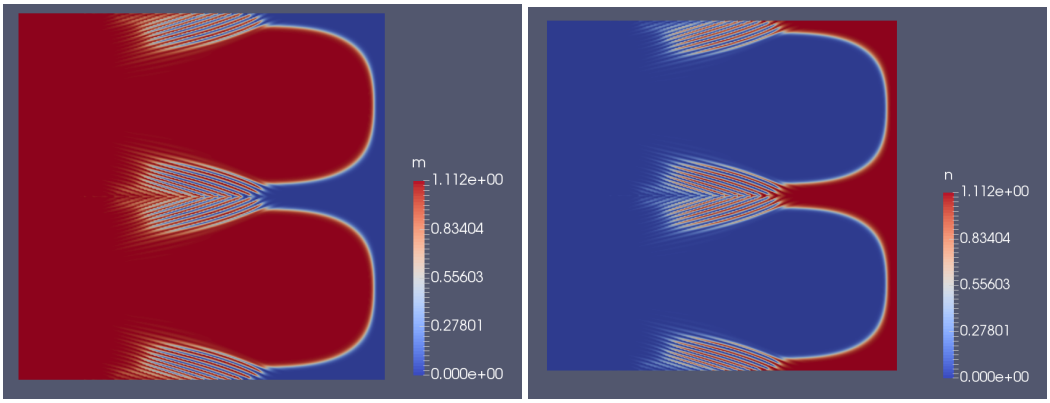


Figure 2.3: Distribution of the concentration of cancerous cells (sx) and of the healthy ones (dx), at time $t = 0.90$. We are in the case $\mu < \nu$ and the concentrations are quite separated.

guess and condition;

- *Prepare lhs and rhs for solver*: here it is implemented the use of method `assemble ()` of the distributed classes addressed in Chapter 2, that prepares the parallel division of lhs and rhs as solver LIS requires;

- *Solve*: it solves the linear system of lhs and rhs.

Figure 3.2 reports the trend of execution time respect to the number of processors. We had to plot the graph of the assembling step in a separate

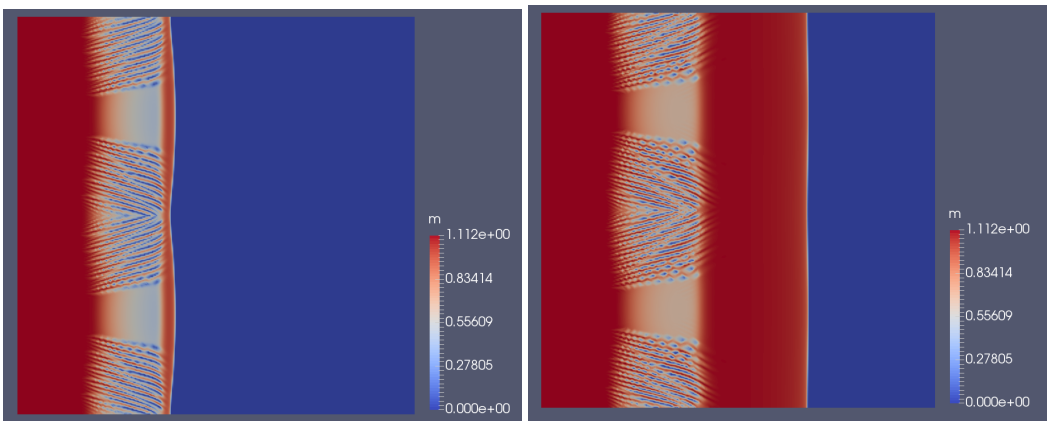
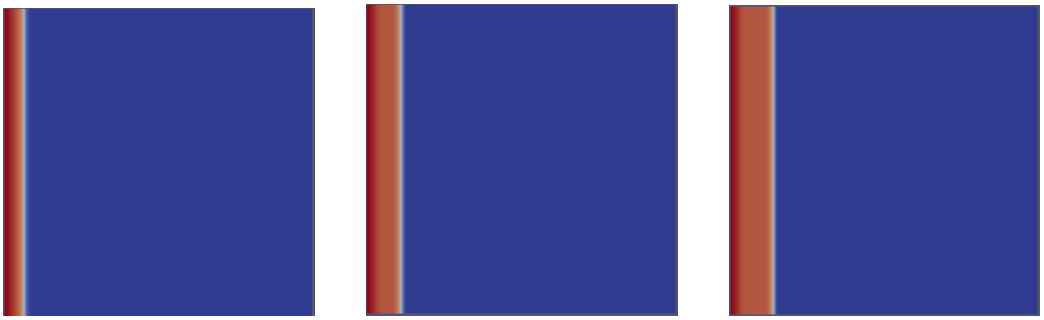


Figure 2.4: Wave of cancerous cells at time $t = 0.135$ (sx) and time $t = 0.27$ (dx) with $\mu > \nu$. As in the one-dimensional case, the concentration of healthy cells is left behind the propagation of cancerous cells, that is faster than with $\mu < \nu$.



(a) With 1 processor: T = 0.06

(b) With 2 processors: T = 0.14

(c) With 4 processors: T = 0.22

window, because its times are much bigger; indeed the assembly of lhs and rhs takes more the 70% of the total computational time.

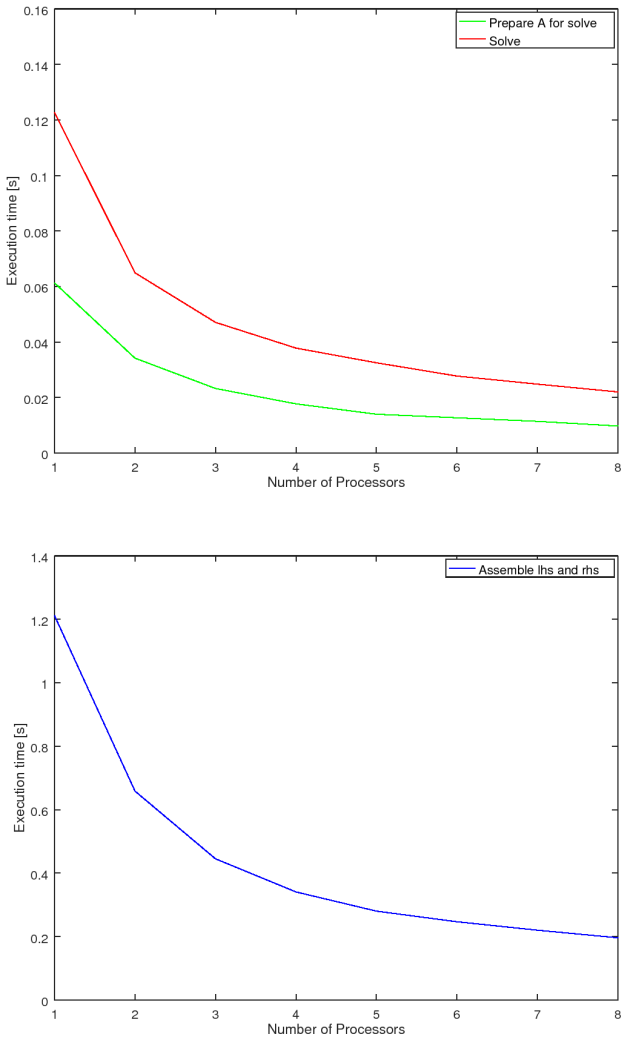


Figure 3.2: Trend of computational time respect to the number of processors.

Bibliography

[1] Peter N Brown and Youcef Saad. Hybrid krylov methods for nonlinear systems of equations. *SIAM Journal on Scientific and Statistical Computing*, 11(3):450–481, 1990.

[2] Carsten Burstedde, Lucas C Wilcox, and Omar Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.

[3] Paul H Calamai and Jorge J Moré. Projected gradient methods for linearly constrained problems. *Mathematical programming*, 39(1):93–116, 1987.

[4] Jinhai Chen and Cornelis Vuik. Globalization technique for projected newton–krylov methods. *International Journal for Numerical Methods in Engineering*, 110(7):661–674, 2017.

[5] Emil Catinaş. The inexact, inexact perturbed, and quasi-newton methods are equivalent models. *Mathematics of computation*, 74(249):291–301, 2005.

[6] Stanley C Eisenstat and Homer F Walker. Choosing the forcing terms in an inexact newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, 1996.

[7] Eli M Gafni and Dimitri P Bertsekas. Two-metric projection methods for constrained optimization. *SIAM Journal on Control and Optimization*, 22(6):936–964, 1984.

[8] Eli M Gafni, Dimitri P Bertsekas, et al. Convergence of a gradient projection method. 1982.

[9] E Hairer, SP Norsett, and G Wanner. Solving ordinary differential equations i, computational mathematics, vol. 8, 1987.

[10] Carl T Kelley. *Solving nonlinear equations with Newton's method*, volume 1. Siam, 2003.

[11] Tommaso Lorenzi, Alexander Lorz, and Benoît Perthame. On interfaces between cell populations with different mobilities. *Kinetic and Related Models*, 10(1):299–311, 2017.

[12] Michael JD Powell. A hybrid method for nonlinear equations. *Numerical methods for nonlinear algebraic equations*, 1970.

[13] Youcef Saad and Martin H Schultz. Gmres: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal on scientific and statistical computing*, 7(3):856–869, 1986.

[14] LK Schubert. Modification of a quasi-newton method for nonlinear equations with a sparse jacobian. *Mathematics of Computation*, 24(109):27–30, 1970.

[15] S van Veldhuizen, Cornelis Vuik, and Chris R Kleijn. On projected newton–krylov solvers for instationary laminar reacting gas flows. *Journal of Computational Physics*, 229(5):1724–1738, 2010.