Numerical Methods Library for OCTAVE

INTERNSHIP'S REPORT

Lilian Calvet

October 21, 2008

Contents

1 Introduction

The Polytechnic Institute of Bragança is pointed as a reference in the scenery of the Portuguese Polytechnics, due to the results achieved through continuous efforts on a dynamic and qualified education. I realise an internship this summer 2008 in its Mathematics Department with my supervisor Carlos Balsa.

I had to execute a Numerical Methods Library for Octave (open source) bound to be used by Bac+2 and Bac+3 students in Mathematics Department. GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language.

The library's content allows students to access to a large pannel of most useful functions in scientific programming. It can also be used as a teaching support with its Documentation and User's Guide. In addition, Mr Carlos Balsa wanted to associate this project with an interactive content and a simulation program realised by my colleague Floriant Chatre.

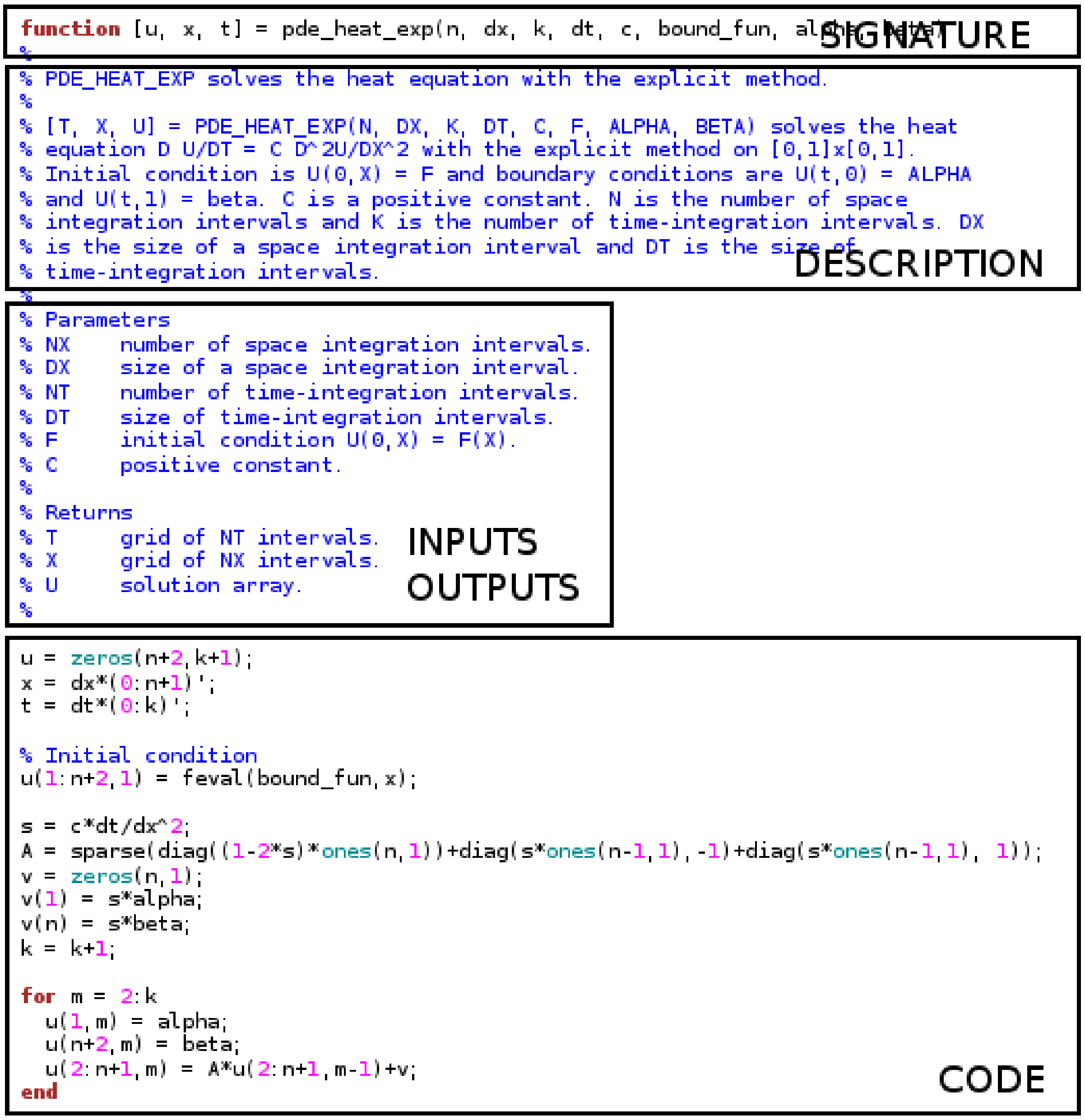
2 Schedule

In this section I present how I managed my work and the different tasks executed :

1. What is Octave ? How to program functions in Octave ?...
2. Listing of what functions already exist in Octave. What are their name? How do we use them ?...
3. Decision about functions' name, variables' name, HELP's appearence...
4. Implementation of each functions from the less complex to the most complex functions. For each function I realised several scripts to check their validity.
5. Realization or complement of the functions'documentation.
6. Implementation of functions to write XML files.
7. Discussion with Mr Chatre Floriant and Mr Carlos Balsa about what results are interesting to display in the simulation program.
8. Creation of the User's Guide.

2.1 An example of a function : PDE_HEAT_EXP ...

Here there is an example of a function ('pde_heat_exp' contained in PDE) and its description :



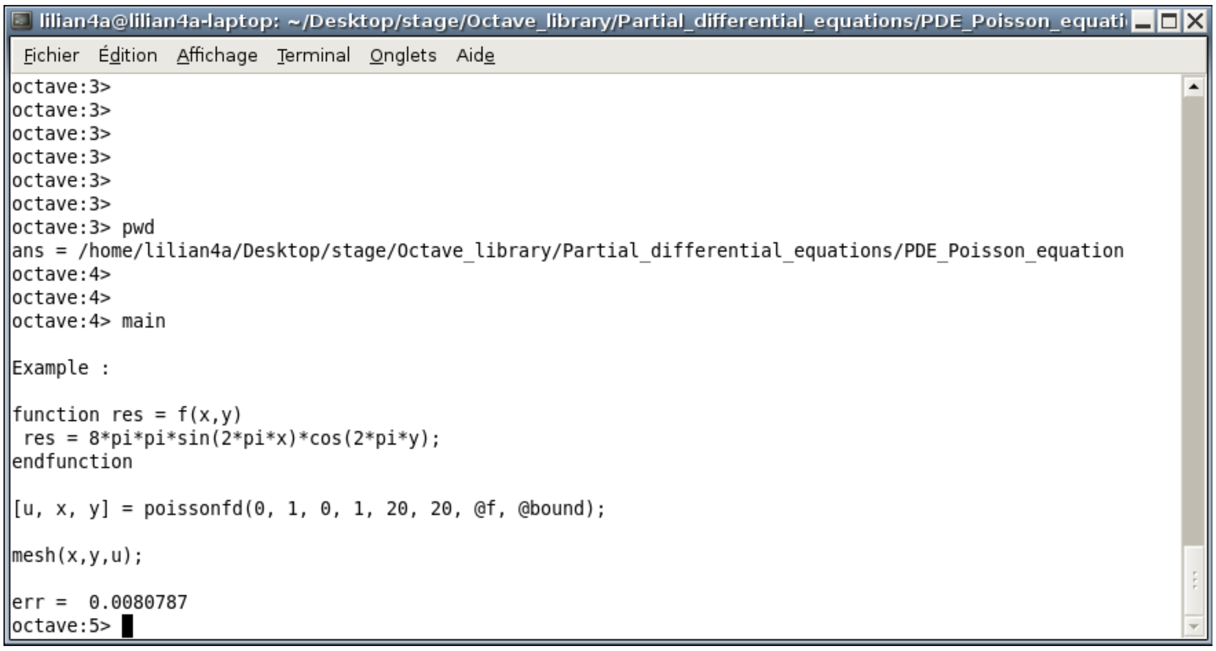
```
function [u, x, t] = pde_heat_exp(n, dx, k, dt, c, bound_fun, alpha, beta)
%
% PDE_HEAT_EXP solves the heat equation with the explicit method.
%
% [T, X, U] = PDE_HEAT_EXP(N, DX, K, DT, C, F, ALPHA, BETA) solves the heat
% equation D U/DT = C D^2U/DX^2 with the explicit method on [0,1]x[0,1].
% Initial condition is U(0,X) = F and boundary conditions are U(t,0) = ALPHA
% and U(t,1) = beta. C is a positive constant. N is the number of space
% integration intervals and K is the number of time-integration intervals. DX
% is the size of a space integration interval and DT is the size of
% time-integration intervals.
%
% Parameters
% NX     number of space integration intervals.
% DX     size of a space integration interval.
% NT     number of time-integration intervals.
% DT     size of time-integration intervals.
% F      initial condition U(0,X) = F(X).
% C      positive constant.
%
% Returns
% T      grid of NT intervals.
% X      grid of NX intervals.
% U      solution array.
%
u = zeros(n+2,k+1);
x = dx*(0:n+1)';
t = dt*(0:k)';

% Initial condition
u(1:n+2,1) = feval(bound_fun,x);

s = c*dt/dx^2;
A = sparse(diag((1-2*s)*ones(n,1))+diag(s*ones(n-1,1),-1)+diag(s*ones(n-1,1), 1));
v = zeros(n,1);
v(1) = s*alpha;
v(n) = s*beta;
k = k+1;

for m = 2:k
  u(1,m) = alpha;
  u(n+2,m) = beta;
  u(2:n+1,m) = A*u(2:n+1,m-1)+v;
end
```

Figure 1: Function description with PDE_HEAT_EXP

2.2 What does the script calling PDE_POISSON_EXP produce ?

Next there is the execution of the script allowing me to validate the function and/or showing how to use the function.

What is the the result of the script 'main' in PDE_Poisson_equation ?



```
octave:3>
octave:3>
octave:3>
octave:3>
octave:3>
octave:3>
octave:3> pwd
ans = /home/lilian4a/Desktop/stage/Octave_library/Partial_differential_equations/PDE_Poisson_equation
octave:4>
octave:4>
octave:4> main

Example :

function res = f(x,y)
 res = 8*pi*pi*sin(2*pi*x)*cos(2*pi*y);
endfunction

[u, x, y] = poissonfd(0, 1, 0, 1, 20, 20, @f, @bound);

mesh(x,y,u);

err =  0.0080787
octave:5>
```

lilian4a@lilian4a-laptop: ~/Desktop/stage/Octave_library/Partial_differential_equations/PDE_Poisson_equati
Fichier
Édition
Affichage
Terminal
Onglets
Aide

Figure 2: What do we obtain with the script executing PDE_POISSON_EXP ?

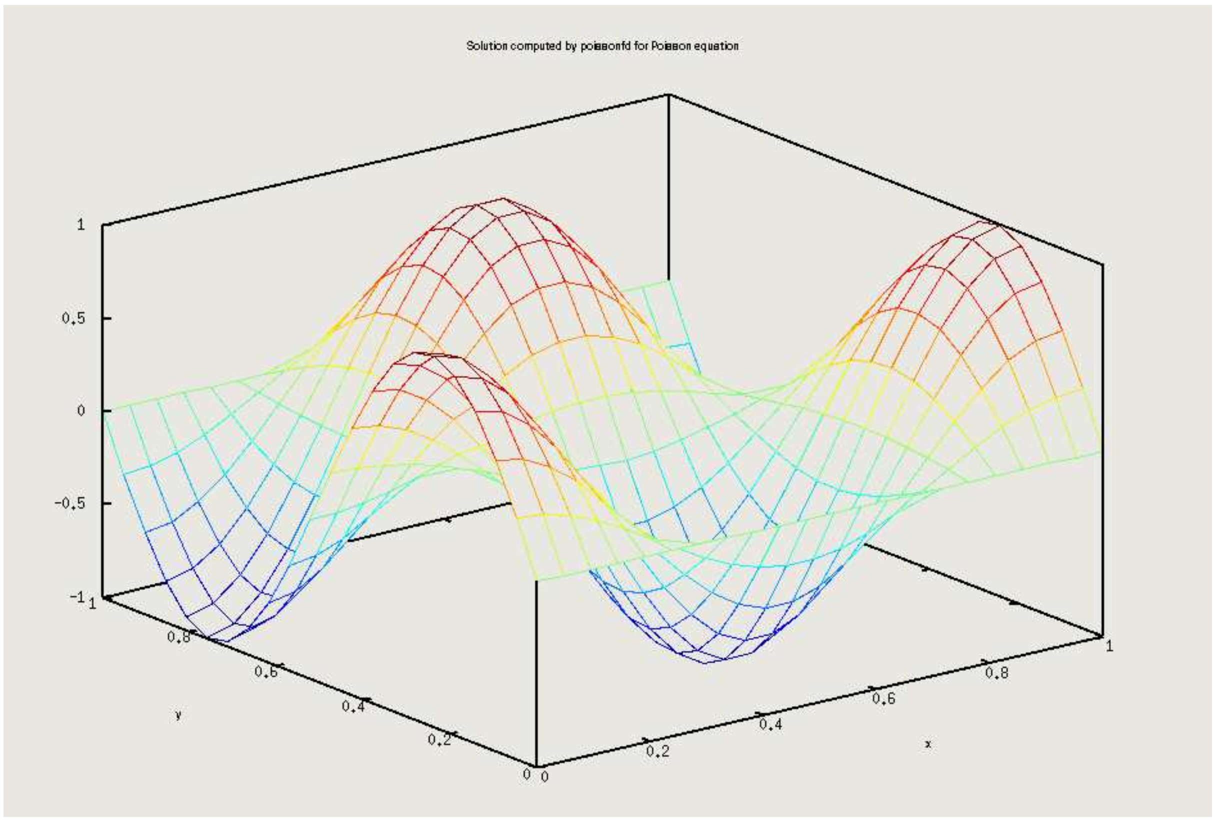


Figure 3: Result of the script executing PDE_POISSON_EXP

2.3 About the User's guide

Here is the content of the User's guide :

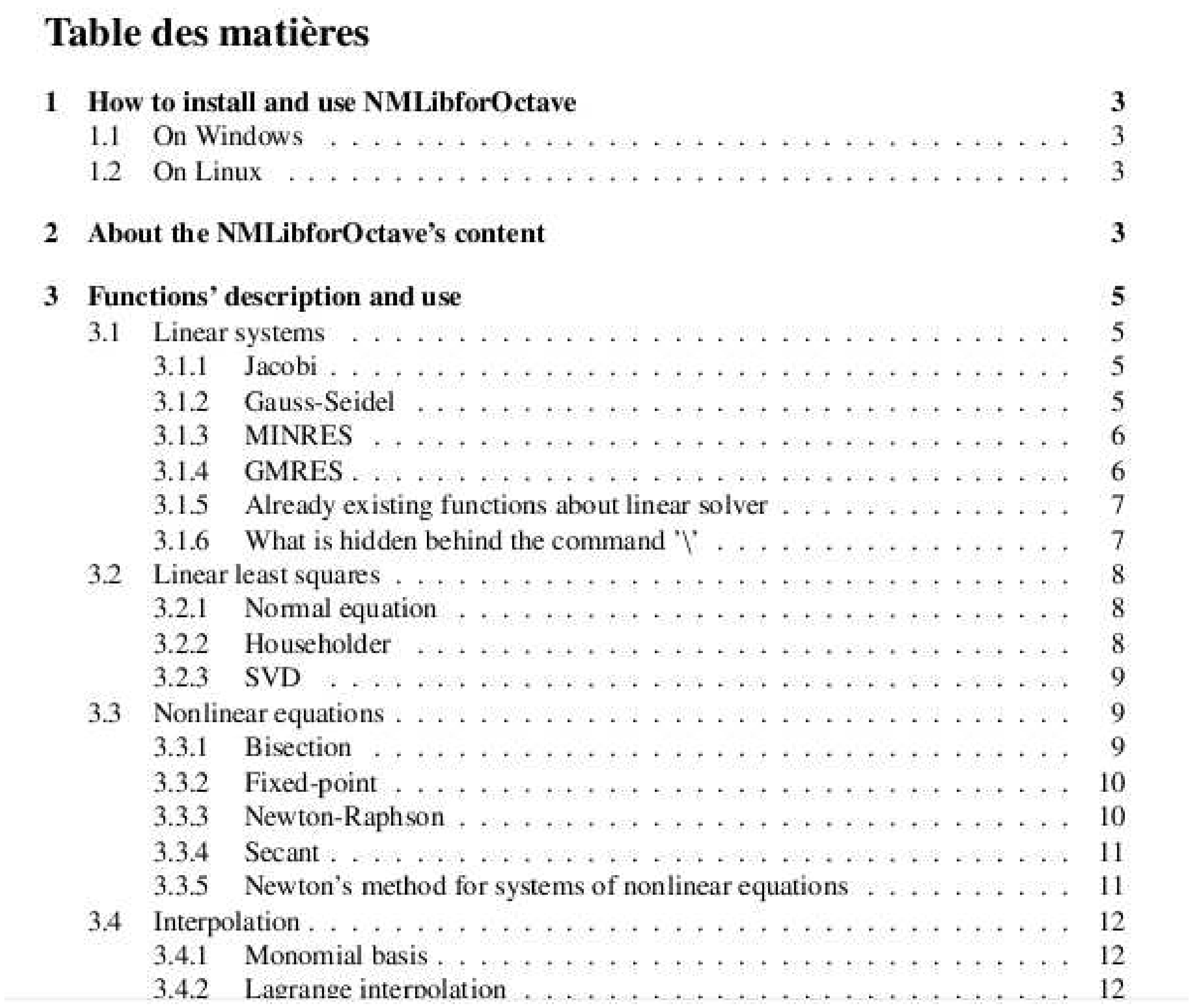
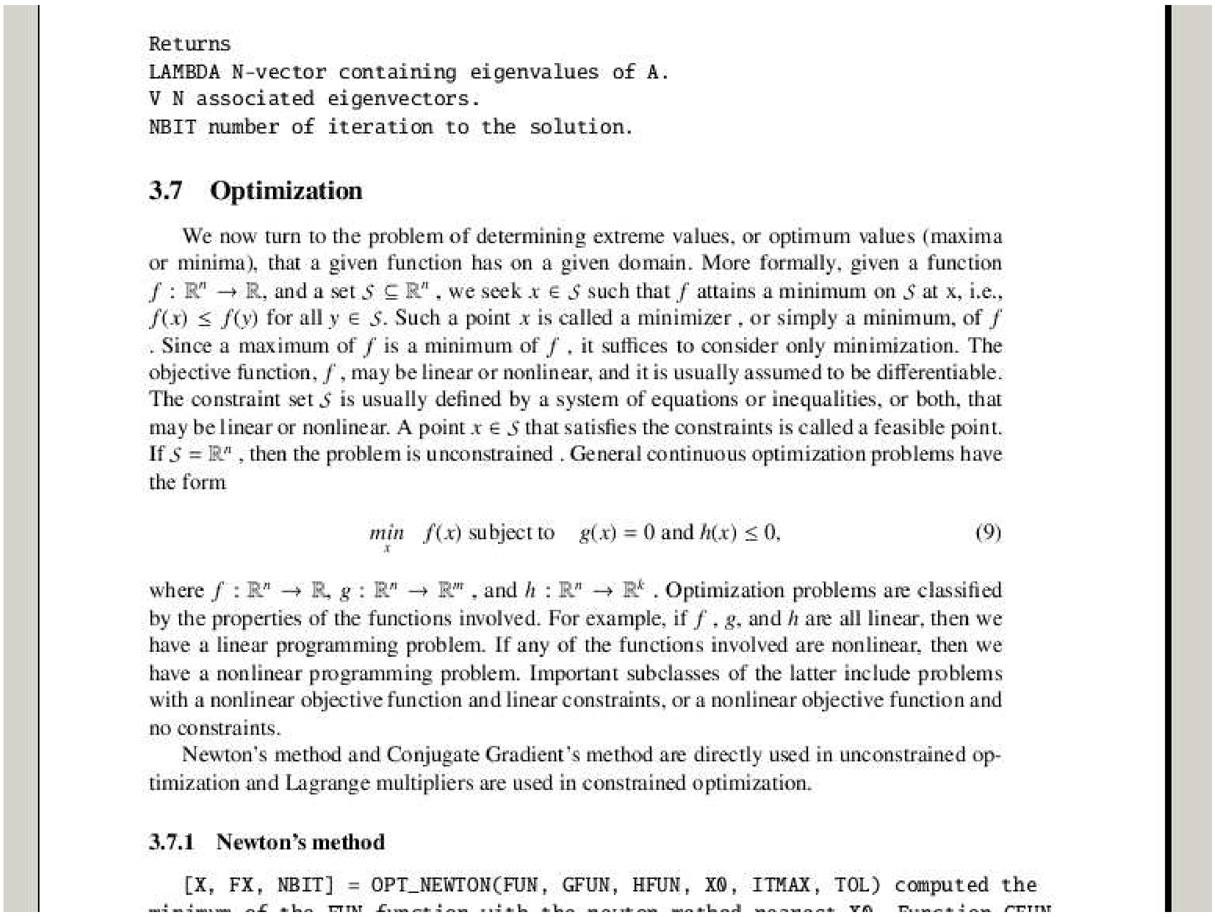


Table des matières

Figure 4: A part of the Table of Contents

In the each subsection of 'Functions description and use' there is an introduction which can describes application domain, to which are destinated each functions...



```
Returns
LAMBDA N-vector containing eigenvalues of A.
V N associated eigenvectors.
NBIT number of iteration to the solution.
```

3.7 Optimization

We now turn to the problem of determining extreme values, or optimum values (maxima or minima), that a given function has on a given domain. More formally, given a function $f : \mathbb{R}^n \to \mathbb{R}$, and a set $S \subseteq \mathbb{R}^n$, we seek $x \in S$ such that f attains a minimum on S at x, i.e., $f(x) \leq f(y)$ for all $y \in S$. Such a point x is called a minimizer , or simply a minimum, of f . Since a maximum of f is a minimum of f , it suffices to consider only minimization. The objective function, f , may be linear or nonlinear, and it is usually assumed to be differentiable. The constraint set S is usually defined by a system of equations or inequalities, or both, that may be linear or nonlinear. A point $x \in S$ that satisfies the constraints is called a feasible point. If $S = \mathbb{R}^n$, then the problem is unconstrained . General continuous optimization problems have the form

$$\min_x \quad f(x) \text{ subject to } \quad g(x) = 0 \text{ and } h(x) \leq 0, \tag{9}$$

where $f : \mathbb{R}^n \to \mathbb{R}$, $g : \mathbb{R}^n \to \mathbb{R}^m$, and $h : \mathbb{R}^n \to \mathbb{R}^k$. Optimization problems are classified by the properties of the functions involved. For example, if f , g, and h are all linear, then we have a linear programming problem. If any of the functions involved are nonlinear, then we have a nonlinear programming problem. Important subclasses of the latter include problems with a nonlinear objective function and linear constraints, or a nonlinear objective function and no constraints.

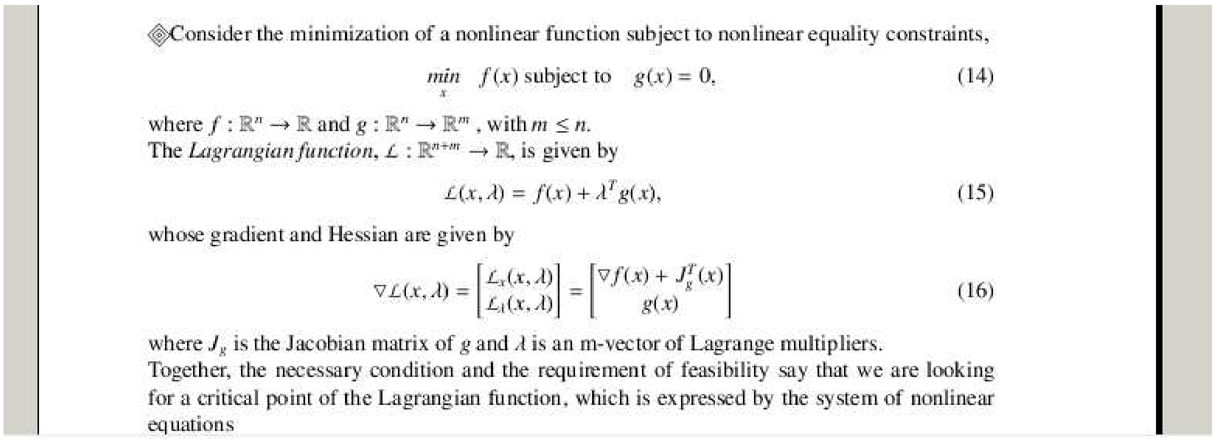
Newton's method and Conjugate Gradient's method are directly used in unconstrained optimization and Lagrange multipliers are used in constrained optimization.

3.7.1 Newton's method

`[X, FX, NBIT] = OPT_NEWTON(FUN, GFUN, HFUN, X0, ITMAX, TOL)` computed the

Figure 5: An example with 'Optimization'

Moreover I add some informations which are really necessary to use the function...with the symbol ◈.



◈Consider the minimization of a nonlinear function subject to nonlinear equality constraints,

$$\min_x \quad f(x) \text{ subject to } \quad g(x) = 0, \tag{14}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $g : \mathbb{R}^n \to \mathbb{R}^m$, with $m \leq n$.
The *Lagrangian function*, $\mathcal{L} : \mathbb{R}^{n+m} \to \mathbb{R}$, is given by

$$\mathcal{L}(x, \lambda) = f(x) + \lambda^T g(x), \tag{15}$$

whose gradient and Hessian are given by

$$\nabla\mathcal{L}(x, \lambda) = \begin{bmatrix} \mathcal{L}_x(x, \lambda) \\ \mathcal{L}_\lambda(x, \lambda) \end{bmatrix} = \begin{bmatrix} \nabla f(x) + J_g^T(x) \\ g(x) \end{bmatrix} \tag{16}$$

where J_g is the Jacobian matrix of g and λ is an m-vector of Lagrange multipliers.
Together, the necessary condition and the requirement of feasibility say that we are looking for a critical point of the Lagrangian function, which is expressed by the system of nonlinear equations

Figure 6: Remark about Lagrange multipliers' method

3 Library's content

3.1 Linear systems

3.1.1 Jacobi

[X, RES, NBIT] = JACOBI(A,B,X0,ITMAX,TOL) computes the solution of the linear system A*X = B with the Jacobi's method. If JACOBI fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
A a square matrix.
B right hand side vector.
X0 initial point.
ITMAX maximum number of iteration.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
RES norm of the residual in X solution.
NBIT number of iterations to compute X solution.

3.1.2 Gauss-Seidel

[X, RES, NBIT] = GAUSS_SEIDEL(A,B,X0,ITMAX,TOL) computes the solution of the linear system A*X = B with the Gauss-Seidel's method. If GAUSS_SEIDEL fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
A a square matrix.
B right hand side vector.
X0 initial point.
ITMAX maximum number of iteration.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
RES norm of the residual in X solution.
NBIT number of iterations to compute X solution.

3.1.3 MINRES

[X,FLAG,RELRES,ITN,RESVEC] = MINRES(A,B,RTOL,MAXIT) solves the linear system of equations A*X = B by means MINRES iterative method.

Parameters
A square (preferably sparse) matrix. In principle A should be symmetric.
B right hand side vector.
TOL relative tolerance for the residual error.
MAXIT maximum allowable number of iterations.
X0 initial guess.

Returns
X computed approximation to the solution.
RELRES ratio of the final residual to its initial value, measured in the Euclidean norm.
ITN actual number of iterations performed.
RESVEC describes the convergence history of the method.

3.1.4 GMRES

[X, NBIT, BCK_ER, FLAG] = GMRES(A,B,X0,ITMAX,M,TOL,M1,M2,LOCATION) attempts to solve the system of linear equations A*x = b for x. The n-by-n coefficient matrix A must be square and should be large and sparse. The column vector B must have length n. If GMRES fails to converge after the maximum number of iterations or halts for any reason, a message is displayed. GMRES restarts all m-iterations.

Parameters
A a square matrix.
B right hand side vector.
X0 initial guess.
ITMAX maximum number of iteration.
M GMRES restarts all m-iterations.
TOL tolerance on the stopping criterion.
M1,M2 preconditionners
LOCATION preconditionners' location : if left then location=1 else right and location=2.

Returns
X computed solution.
NBIT number of iterations to compute X solution.
BCK_ER describes the convergence history of the method.
FLAG if the method converges then FLAG=0 else FLAG=-1.

3.1.5 Already existing functions about linear solver

It already exists function to solve linear systems in Octave. We have particularly the Conjugate Gradient method *pcg*, the Cholesky factorization *chol* and finally LU factorization *lu*.

3.2 Linear least squares

3.2.1 Normal equation

[X, RES] = NORMALEQ(A,B) computes the solution of linear least squares problem min(norm(A*X-B,2)) solving associated normal equation A'*A*X = A'*B.

Parameters
A a matrix.
B right hand side vector.
Returns

X computed solution.
RES value of norm(A*X-B) with X solution computed.

3.2.2 Householder

[X, RES] = LLS_HQR(A,B) computes the solution of linear least squares problem min(norm(A*X-B,2)) using Householder QR.

Parameters

A a matrix.
B right hand side vector.

Returns
X computed solution.
RES value of norm(A*X-B) with X solution computed.

3.2.3 SVD

[X, RES] = SVD_LEAST_SQUARES(A,B) computes the solution of linear least squares problem min(norm(A*X-B,2)) using the singular value decomposition of A.

Parameters
A a matrix.
B right hand side vector.

Returns
X computed solution.
RES value of norm(A*X-B) with X solution computed.

3.3 Nonlinear equations

3.3.1 Bisection

X = BISECTION(FUN,A,B,ITMAX,TOL) tries to find a zero X of the continuous function FUN in the interval [A, B] using the bisection method. FUN accepts real scalar input x and returns a real scalar value. If the search fails an error message is displayed. FUN can also be an inline object.

[X, RES, NBIT] = BISECTION(FUN,A,B,ITMAX,TOL) returns the value of the residual in X solution and the iteration number at which the solution was computed.

Parameters
FUN evaluated function.
A,B [A,B] interval where the solution is computed, A < B and sign(FUN(A)) = - sign(FUN(B)).
ITMAX maximal number of iterations.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
NBIT number of iterations to find the solution.
RES value of the residual in X solution.

3.3.2 Fixed-point

X = FIXED_POINT(FUN,X0,ITMAX,TOL) solves the scalar nonlinear equation such that ’FUN(X) == X’ with FUN continuous function. FUN accepts real scalar input X and returns a real scalar value. If the search fails an error message is displayed. FUN can also be inline objects.

[X, RES, NBIT] = FIXED_POINT(FUN,X0,ITMAX,TOL) returns the norm of the residual in X solution and the iteration number at which the solution was computed.

Parameters
FUN evaluated function.
DFUN f’s derivate.
X0 initial point.
ITMAX maximal number of iteration.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
RES norm of the residual FUN(X)-X in X solution.
NBIT number of iterations to find the solution.

3.3.3 Newton-Raphson

X = NLE_NEWTRAPH(FUN,DFUN,X0,ITMAX,TOL) tries to find a zero X of the continuous and differentiable function FUN nearest to X0 using the Newton-Raphson method. FUN and its derivate DFUN accept real scalar input x and returns a real scalar value. If the search fails an error message is displayed. FUN and DFUN can also be inline objects.

[X, RES, NBIT] = NLE_NEWTRAPH(FUN,DFUN,X0,ITMAX,TOL) returns the value of the residual in X solution and the iteration number at which the solution was computed.

Parameters
FUN evaluated function.
DFUN f’s derivate.
X0 initial point.
ITMAX maximal number of iterations.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
RES value of the residual in x solution.
NBIT number of iterations to find the solution.

3.3.4 Secant

X = SECANT(FUN,X1,X2,ITMAX,TOL) tries to find a zero X of the continuous function FUN using the secant method with starting points X1, X2. FUN accepts real scalar input X and returns a real scalar value. If the search fails an error message is displayed. FUN can also be an inline object.

[X, RES, NBIT] = SECANT(FUN,X1,X2,ITMAX,TOL) returns the value of the residual in X solution and the iteration number at which the solution was computed.

Parameters
FUN evaluated function.
X1,X2 starting points.
ITMAX maximal number of iterations.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
RES value of the residual in X solution.
NBIT number of iterations to find the solution.

3.3.5 Newton's method for systems of nonlinear equations

X = NLE_NEWTSYS(FFUN,JFUN,X0,ITMAX,TOL) tries to find the vector X, zero of a nonlinear system defined in FFUN with jacobian matrix defined in the function JFUN, nearest to the vector X0.

[X, RES, NBIT] = NLE_NEWTSYS(FUN,DFUN,X0,ITMAX,TOL) returns the norm of the residual in X solution and the iteration number at which the solution was computed.

Parameters
FFUN evaluated function.
JFUN FFUN's jacobian matrix.
X0 initial point.
ITMAX maximal number of iterations.
TOL tolerance on the stopping criterion.

Returns
X computed solution.
RES norm of the residual in X solution.
NBIT number of iterations to find the solution.

3.4 Interpolation

3.4.1 Monomial basis

P = ITPOL_MONOM(X,Y,x) computes the monomial basis interpolation of points defined by x-coordinate X and y-coordinate Y. x can be a real vector, each row in the solution array P corresponds to a x-coordinate in the vector x.

Parameters
X abscissas of interpolated points.
Y odinates of interpolated points.
x can be a scalar or a vector of values.

Returns
P value of p(x).

3.4.2 Lagrange interpolation

P = LAGRANGE(X,Y,x) computes the polynomial Lagrange interpolation of points defined by x-coordinate X and y-coordinate Y. x can be a real vector, each row in the solution array P corresponds to a x-coordinate in the vector x.

Parameters
X abscissas of interpolated points.
Y odinates of interpolated points.
x can be a scalar or a vector of values.

Returns
P value of P(x).

3.4.3 Newton interpolation

P = ITPOL_NEWT(X,Y,x) computes the polynomial Newton interpolation of points defined by x-coordinate X and y-coordinate Y. x can be a real vector, each row in the solution array P corresponds to a x-coordinate in the vector x.

Parameters
X abscissas of interpolated points.
Y odinates of interpolated points.
x can be a scalar or a vector of values.

Returns
P value of p(x).

3.5 Numerical integration

3.5.1 Trapezoid's rule

RES = INTE_TRAPEZ(FUN,A,B,N) computes an approximation of the integral of the function FUN via the trapezoid method (using N equispaced intervals). FUN accepts real scalar input x and returns a real scalar value. FUN can also be an inline object.

Parameters
FUN integrated function.
A,B FUN is integrated on [A,B].
N number of subdivisions.

Returns
RES result of integration.

3.5.2 Simpson's rule

RES = INTE_SIMPSON(FUN,A,B,N) computes an approximation of the integral of the function FUN via the Simpson method (using N equispaced intervals). FUN accepts real scalar input x and returns a real scalar value. FUN can also be an inline object.

Parameters
FUN integrated function.
A,B FUN is integrated on [A,B].
N number of subdivisions.

Returns
RES result of integration.

3.5.3 Newton-Cotes' rule

RES = INTE_NEWTCOT(FUN,A,B,N) computes an approximation of the integral of the function FUN via the Newton-Cotes method (using N equispaced intervals). FUN accepts real scalar input x and returns a real scalar value. FUN can also be an inline object.

Parameters
FUN integrated function.
A,B FUN is integrated on [A,B].
N number of subdivisions.

Returns RES result of integration.

3.6 Eigenvalue problems

3.6.1 Power iteration

[LAMBDA, V, NBIT] = EIG_POWER(A, X0, ITMAX, TOL) computes dominant eigenvalue and associated eigenvector of A with power iteration method. If EIG_POWER fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
A a square matrix.
X0 initial point.
ITMAX maximal number of iterations.
TOL maximum relative error.

Returns
LAMBDA dominant eigenvalue of A.
V associated eigenvector.
NBIT number of iteration to the solution.

3.6.2 Inverse method

[LAMBDA, V, NBIT] = EIG_INVERSE(A, X0, ITMAX, TOL) Compute the smallest eigenvalue of A and associated eigenvector with inverse method. If EIG_INVERSE fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
A a square matrix.
X0 initial point.
ITMAX maximal number of iterations.
TOL maximum relative error.

Returns
LAMBDA smallest eigenvalue of A.
V associated eigenvector.
NBIT number of iteration to the solution.

3.6.3 Rayleigh quotient iteration

[LAMBDA, V, NBIT] = EIG_RAYLEIGH(A, X0, ITMAX, TOL) computes the best estimate of an eigenvalue of A associated to an approximate eigenvector X0 with Rayleigh quotient iteration method. If EIG_RAYLEIGH fails to converge after the maximum number of iterations or halts for any reason, a message is displayed. This method can be used to accelerate the convergence of a method such as power iteration.

Parameters
A a square matrix.
X0 initial point corresponding to an approximate eigenvector.
ITMAX maximal number of iterations.
TOL maximum relative error.

Returns
LAMBDA the best estimate for the corresponding eigenvalue.
V associated eigenvector.
NBIT number of iteration to the solution.

3.6.4 Orthogonal iteration

[LAMBDA, V, NBIT] = EIG_ORTHO(A, X0, ITMAX, TOL) computes P=size(X0,2) eigenvalues and associated eigenvectors of A with orthogonal iteration method. If EIG_ORTHO fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
A a square matrix.
X0 arbitrary N x P matrix of rank P, contains X0(1),X0(2),...X0(P)
linearly independant.
ITMAX maximal number of iterations.
TOL maximum relative error.

Returns
LAMBDA P-vector containing eigenvalues of A.
V P eigenvectors.
NBIT number of iteration to the solution.

3.6.5 QR iteration

[LAMBDA, V, NBIT] = EIG_QR(A, ITMAX, TOL) computes N (=size(A)) eigenvalues and associated eigenvectors of A with orthogonal iteration method. If EIG_QR fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
A (N*N) matrix
ITMAX maximal number of iterations.
TOL maximum relative error.

Returns
LAMBDA N-vector containing eigenvalues of A.
V N associated eigenvectors.

NBIT number of iteration to the solution.

3.7 Optimization

3.7.1 Newton's method

[X, FX, NBIT] = OPT_NEWTON(FUN, GFUN, HFUN, X0, ITMAX, TOL) computed the minimum of the FUN function with the newton method nearest X0. Function GFUN defines the gradient vector and function HFUN defines the hessian matrix. FUN accepts a real vector input and return a real vector. FUN, GFUN and HFUN can also be inline object. If OPT_NEWTON fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
FUN evaluated function.
GFUN FUN's gradient function.
HFUN FUN's hessian matrix function.
X0 initial point.
ITMAX maximal number of iterations.
TOL tolerance on the stopping criterion.

Returns
X computed solution of min(FUN).
FX value of FUN(X) with X computed solution.
NBIT number of iterations to find the solution.

3.7.2 Conjugate gradient method

[X, FX, NBIT] = OPT_CG(FUN, X0, GFUN, HFUN, TOL, ITMAX) computed the minimum of the FUN function with the conjugate gradient method nearest X0. Function GFUN defines gradient vector and function HFUN defines hessian matrix. FUN accepts a real vector input and return a real vector. FUN, GFUN and HFUN can also be inline object. If OPT_CG fails to converge after the maximum number of iterations or halts for any reason, a message is displayed.

Parameters
FUN evaluated function.
X0 initial point.
GFUN FUN's gradient function.
HFUN FUN's hessian matrix function.
TOL tolerance on the stopping criterion.
ITMAX maximal number of iterations.

Returns
X computed solution of min(FUN).

FX value of FUN(X) with X computed solution.
NBIT number of iterations to find the solution.

3.7.3 Lagrange multipliers

[XMIN, LAMBDAMIN, FMIN] = OPT_LAGRANGE(F, GRADF, G, JACG, X0) computed the minimum of the function FUN subject to ’G(X) = 0’ with the lagrange multiplier method. Function GRADF defines the gradient vector of F. Function G represents equality-constrained and function JACG defines its jacobian matrix. F and G accept a real vector input and return a real vector. F, GRADF, G and JACG can also be inline object.

Parameters
F evaluated function.
GRADF F’s gradient function.
G equality-constrained function : ’G(X) = 0’.
X0 initial point.

Returns
XMIN computed solution of min(FUN).
LAMBDAMIN vector of Lagrange multipliers on XMIN.
FX value of FUN(X) with X computed solution.

3.8 Initial value problems for Ordinary differential Equations

3.8.1 Euler

[TT,Y] = ODE_EULER(ODEFUN,TSPAN,Y,NH) with TSPAN = [T0, TF] integrates the system of differential equations Y’=f(T,Y) from time T0 to TF with initial condition Y0 using the forward Euler method on an equispaced grid of NH intervals. Function ODEFUN(T,Y) must return a column vector corresponding to f(T, Y). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters
ODEFUN integrated function.
TSPAN TSPAN = [T0 TF]
Y initial value Y(T0).
NH TT equispaced grid of NH intervals.

Returns
TT equispaced grid of NH intervals.
Y solution array.

3.8.2 Implicit Euler

[TT,Y] = ODE_BEULER(ODEFUN,TSPAN,Y,NH) with TSPAN = [T0, TF] integrates the system of differential equations Y'=f(T,Y) from time T0 to TF with initial condition Y0 using the backward Euler method on an equispaced grid of NH intervals. Function ODEFUN(T, Y) must return a column vector corresponding to f(T, Y). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters
ODEFUN integrated function.
TSPAN TSPAN = [T0 TF].
Y initial value Y(T0).
NH TT equispaced grid of NH intervals.

Returns
TT equispaced grid of NH intervals.
Y solution array.

3.8.3 Modified Euler

[TT,Y] = ODE_EULER(ODEFUN,TSPAN,Y,NH) with TSPAN = [T0, TF] integrates the system of differential equations Y'=f(T,Y) from time T0 to TF with initial condition Y0 using the modified Euler method on an equispaced grid of NH intervals. Function ODEFUN(T,Y) must return a column vector corresponding to f(T, Y). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters
ODEFUN integrated function.
TSPAN TSPAN = [T0 TF].
Y initial value Y(T0).
NH TT equispaced grid of NH intervals.

Returns
TT equispaced grid of NH intervals.
Y solution array.

3.8.4 Fourth-order Rounge-Kutta

It corresponds to ode23,ode45 which already exit in Octave.

3.8.5 Fourth-order predictor

[TT,Y] = ODE_FOP(ODEFUN,TSPAN,Y,NH) with TSPAN = [T0, TF] integrates the system of differential equations Y'=f(T,Y) from time T0 to TF with initial condition Y0 using the fourth-order predictor scheme on an equispaced grid of NH intervals. Function ODEFUN(T,Y) must return a column vector corresponding to f(T, Y). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters
ODEFUN integrated function.
TSPAN tspan = [T0 TF].
Y initial value Y(T0).
NH TT equispaced grid of NH intervals.

Returns
TT equispaced grid of NH intervals.
Y solution array.

3.9 Boundary value problems for Ordinary differential Equations

3.9.1 Shooting method

[T,Y] = ODE_SHOOT(IVP, A, B, UA, UB) integrates the system of differential equations u”= f(t,u,u’) from time A to B with boundary conditions u(A) = UA and u(B) = UB. Function IVP(t,u,u’) must return a double column vector [u’, u”] with u”= f(t,u,u’). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters
IVP integrated function.
A T0.
B TF.
UA initial value Y(T0).
UB final value Y(TF).

Returns
T equispaced grid.
Y solution array.

3.9.2 Finite difference method

[T,Y] = ODE_FINIT_DIFF(RHS, A, B, UA, UB, N) integrates the system of differential equations u”= f(t,u,u’) from time A to B with boundary conditions u(A) = UA and u(B) = UB on an equispaced grid of N intervals. Function RHS(t,u,u’) must return a column vector corresponding to f(t,u,u’). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters

RHS integrated function.
A T0.
B TF.
UA initial value Y(T0).
UB final value Y(TF).
N T equispaced grid of N intervals.

Returns
T equispaced grid of N intervals.
Y solution array.

3.9.3 Colocation method

[T,Y] = ODE_COLLOC(RHS, A, B, UA, UB, DN, N) integrates the system of differential equations u”= f(t,u,u’) from time A to B with boundary conditions u(A) = UA and u(B) = UB on an equispaced grid of N intervals. Function RHS(t,u,u’) must return a column vector corresponding to f(t,u,u’). Each row in the solution array Y corresponds to a time returned in the column vector T.

Parameters
RHS integrated function.
A T0.
B TF.
UA initial value Y(T0).
UB final value Y(TF).
N T equispaced grid of N intervals.
DN degree of computed polynomial solution.

Returns
T equispaced grid of N intervals.
Y solution array.

3.10 Partial Differential Equations

3.10.1 Method of lines (for Heat equation)

[T, X, U] = PDE_HEAT_LINES(NX, NT, C, F) solves the heat equation D U/DT = C D2 U/DX2 with the method of lines on [0,1]x[0,1]. Initial condition is U(0,X) = F. C is a positive constant. NX is the number of space integration intervals and NT is the number of time-integration intervals.

Parameters
NX X equispaced grid of NX intervals.
NT T equispaced grid of NX intervals.

C positive constant.

Returns
T equispaced grid of NT intervals.
X equispaced grid of NX intervals.
U solution array.

3.10.2 2-D solver for Advection equation

[T, X, U] = PDE_ADVEC_EXP(N, DX, K, DT, C, F) solves the advection equation D U/DT = -C D U/DX with the explicit method on [0,1]x[0,1]. Initial condition is U(0,X) = F. C is a positive constant. N is the number of space integration intervals and K is the number of time-integration intervals. DX is the size of a space integration interval and DT is the size of time-integration intervals.

Parameters
NX number of space integration intervals.
DX size of a space integration interval.
NT number of time-integration intervals.
DT size of time-integration intervals.
C positive constant.
F initial condition U(0,X) = F(X).

Returns
T grid of NT intervals.
X grid of NX intervals.
U solution array.

[T, X, U] = PDE_ADVEC_IMP(N, DX, K, DT, C, F) solves the advection equation D U/DT = -C D U/DX with the implicit method on [0,1]x[0,1]. Initial condition is U(0,X) = F. C is a positive constant. N is the number of space integration intervals and K is the number of time-integration intervals. DX is the size of a space integration interval and DT is the size of time-integration intervals.

Parameters
NX number of space integration intervals.
DX size of a space integration interval.
NT number of time-integration intervals.
DT size of time-integration intervals.
C positive constant.
F initial condition U(0,X) = F(X).

Returns
T grid of NT intervals.
X grid of NX intervals.
U solution array.

3.10.3 2-D solver for Heat equation

[T, X, U] = PDE_HEAT_EXP(N, DX, K, DT, C, F, ALPHA, BETA) solves the heat equation D U/DT = C D2U/DX2 with the explicit method on [0,1]x[0,1]. Initial condition is U(0,X) = F and boundary conditions are U(t,0) = ALPHA and U(t,1) = beta. C is a positive constant. N is the number of space integration intervals and K is the number of time-integration intervals. DX is the size of a space integration interval and DT is the size of time-integration intervals.

Parameters
NX number of space integration intervals.
DX size of a space integration interval.
NT number of time-integration intervals.
DT size of time-integration intervals.
F initial condition U(0,X) = F(X).
C positive constant.

Returns
T grid of NT intervals.
X grid of NX intervals.
U solution array.

[T, X, U] = PDE_HEAT_IMP(N, DX, K, DT, C, F, ALPHA, BETA) solves the heat equation D U/DT = C D2U/DX2 with the implicit method on [0,1]x[0,1]. Initial condition is U(0,X) = F and boundary conditions are U(t,0) = ALPHA and U(t,1) = beta. C is a positive constant. N is the number of space integration intervals and K is the number of time-integration intervals. DX is the size of a space integration interval and DT is the size of time-integration intervals.

Parameters
NX number of space integration intervals.
DX size of a space integration interval.
NT number of time-integration intervals.
DT size of time-integration intervals.
F initial condition U(0,X) = F(X).
C positive constant.

Returns
T grid of NT intervals.
X grid of NX intervals.
U solution array.

3.10.4 2-D solver for Wave equation

[T, X, U] = PDE_WAVE_EXP(N, DX, K, DT, C, F, G, ALPHA, BETA) solves the wave equation D2U/DT2 = C D2U/DX2 with the explicit method on [0,1]x[0,1]. Initial condition is U(0,X)

= F, D U/DT (0,X)= G(X) and boundary conditions are U(t,0) = ALPHA and U(t,1) = beta. C is a positive constant. N is the number of space integration intervals and K is the number of time-integration intervals. DX is the size of a space integration interval and DT is the size of time-integration intervals.

Parameters
NX number of space integration intervals.
DX size of a space integration interval.
NT number of time-integration intervals.
DT size of time-integration intervals.
F initial condition U(0,X) = F(X).
G initial condition D U/DT (0,X)= G(X).
C positive constant.

Returns
T grid of NT intervals.
X grid of NX intervals.
U solution array.

[T, X, U] = PDE_WAVE_IMP(N, DX, K, DT, C, F, G, ALPHA, BETA) solves the wave equation D2U/DT2 = C D2U/DX2 with the implicit method on [0,1]x[0,1]. Initial condition is U(0,X) = F, D U/DT (0,X)= G(X) and boundary conditions are U(t,0) = ALPHA and U(t,1) = beta. C is a positive constant. N is the number of space integration intervals and K is the number of time-integration intervals. DX is the size of a space integration interval and DT is the size of time-integration intervals.

Parameters
NX number of space integration intervals.
DX size of a space integration interval.
NT number of time-integration intervals.
DT size of time-integration intervals.
F initial condition U(0,X) = F(X).
G initial condition D U/DT (0,X)= G(X).
C positive constant.

Returns
T grid of NT intervals.
X grid of NX intervals.
U solution array.

3.10.5 2-D solver for the Poisson Equation

POISSONFD two-dimensional Poisson solver [U, X, Y] = POISSONFD(A, C, B, D, NX, NY, FUN, BOUND) solves by five-point finite difference scheme the problem -LAPL(U) = FUN in the rectangle (A,B)x(C,D) with Dirichlet boundery conditions U(X,Y)=BOUND(X,Y) for any (X, Y) on the boundery of the rectangle.

[**U, X, Y, ERROR**] **= POISSONFD(A,C,B,D,NX,NY,FUN,BOUND,UEX)** computes also the maximum nodal error ERROR with respect to the exact solution UEX. FUN, BOUND and UEX can be online functions.

Parameters
A, B
C, D rectangle (A,B)x(C,D) where the solution is computed.
NX X equispaced grid of NX intervals.
NY Y equispaced grid of NY intervals.
FUN
BOUND boundary condition.
UEX exact solution.

Returns
U solution array.
X equispaced grid of NX intervals.
Y equispaced grid of NY intervals.
ERROR maximum nodal error ERROR with respect to the exact solution UEX.

4 Conclusion

This summer internship allow me to improve skills :

- communication between my supervisor, my colleague and I.
- analyze of the topics and what was the main objective of my supervisor executing this library.
- management of my work and respect of the timetable to achieve all the aims.
- reviews and complements about my scientific programming knowledge.
- strictness and precision to produce a component which works perfectly.

Before I finish with this report I would like to thank Carlos Balza and the teaching team to follow me during this training period.

5 How to continue

Finally I think my work can be modified and completed to add functions or enrich the documentation...In fact I strive to keep the same variable's name in all functions with a view to modifying or completing them :

- nbit : number of iterations
- err : relative error of the residual
- x, X : vector, matrix
- aux : auxiliary variable

...