ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Implementation of a Distributed System for the Solution of MultiDomain / MultiPhysics Problems

Ανάπτυξη Κατανεμημένου Συστήματος για Επίλυση Προβλημάτων Πολλαπλών - Χωρίων / Πολλαπλών - Φυσικών Μοντέλων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΚΩΝΣΤΑΝΤΙΝΟΣ Κ. ΧΑΛΚΙΑΣ

Επιβλεποντες Καθηγητές: Παναγιώτα Τσομπανοπούλου
Επίκουρη Καθηγήτρια

Εμμανουήλ Βάβαλης
Καθηγητής

Βόλος, Ιούλιος 2013

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ &
ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάπτυξη Κατανεμημένου Συστήματος για Επίλυση Προβλημάτων Πολλαπλών - Χωρίων / Πολλαπλών - Φυσικών Μοντέλων

Διπλωματική Εργασία

ΚΩΝΣΤΑΝΤΙΝΟΣ Κ. ΧΑΛΚΙΑΣ

Επιβλέποντες Καθηγητές: Παναγιώτα Τσομπανοπούλου
Επίκουρη Καθηγήτρια Π.Θ.

Εμμανουήλ Βάβαλης
Καθηγητής Π.Θ.

Εγκρίθηκε από την διμελή εξεταστική επιτροπή την 5 Ιουλίου 2013

....	....
Π. Τσομπανοπούλου	Ε. Βάβαλης
Επίκουρη Καθηγήτρια	Καθηγητής

Διπλωματική Εργασία για την απόκτηση του Διπλώματος του Μηχανικού Η/Υ, Τηλεπικοινωνιών & Δικτύων του Πανεπιστημίου Θεσσαλίας, στα πλαίσια του Προγράμματος Προπτυχιακών Σπουδών του Τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Πανεπιστημίου Θεσσαλίας.

....

ΚΩΝΣΤΑΝΤΙΝΟΣ ΧΑΛΚΙΑΣ

Διπλωματούχος Μηχανικός Η/Υ, Τηλ/νιών

& Δικτύων

© 2013 – All rights reserved

To my family and friends.

Ευχαριστίες

Σε αυτό το σημείο θα ήθελα να ευχαριστήσω όλους τους ανθρώπους που με βοήθησαν και με υποστήριξαν στα χρόνια σπουδών μου.

Αρχικά θα ήθελα να ευχαριστήσω την επιβλέπουσα της διπλωματικής μου εργασίας, Παναγιώτα Τσομπανοπούλου για την εμπιστοσύνη που μου έδειξε να ασχοληθώ με μία τόσο απαιτητική διπλωματική εργασία. Η υποστήριξή της, η καθοδήγησή της και οι ουσιώδεις παρεμβάσεις της με διευκόλυναν να επιτύχω την ολοκλήρωσή της. Το ενδιαφέρον της και οι συζητήσεις που είχαμε μου έδειξε ότι όχι μόνο είχαμε μια άριστη συνεργασία αλλά και ότι αναπτύξαμε μια ουσιαστική ανθρώπινη σχέση. Θα ήθελα επίσης να ευχαριστήσω τον δεύτερο επιβλέποντα της διπλωματικής μου εργασίας κ. Εμμανουήλ Βάβαλη για τις συμβουλές του και την υποστήριξή του καθόλη τη διάρκεια των σπουδών μου.

Ακόμη οφείλω ένα μεγάλο ευχαριστώ στους φίλους μου, την Αλεξία, τον Βασίλη, την Μαριάννα, τον Κώστα, την Αφροδίτη και την Χρυσή για την αγάπη τους και την υποστήριξή τους και την ανοχή τους στις ιδιοτροπίες μου όλα αυτά τα χρόνια σπουδών αλλά και κατά τη διάρκεια της εκπόνησης της διπλωματικής μου εργασίας.

Τέλος, τα εκπαιδευτικά μου επιτεύγματα είναι το αποτέλεσμα της άνευ όρων αγάπης, υποστήριξης, ενθάρρυνσης , συμβουλής και καθοδήγησης που έχω από την οικογένειά μου. Ευχαριστώ τους γονείς μου, Ευγενία και Κυριαζή, που πάντα είναι δίπλα μου να με στηρίζουν , να με βοηθούν με οποιοδήποτε τρόπο και να μου δίνουν δύναμη να συνεχίσω και να προσπαθώ για το καλύτερο δυνατό. Ακόμη θα ήθελα να ευχαριστήσω θερμά τα αδέρφια μου, Μάγδα και Περικλή, για την αγάπη τους και την κατανόησή τους.

Κωστής,
Βόλος, 2013

Περίληψη

Η μοντελοποίηση και η προσομοίωση σύνθετων φυσικών προβλημάτων περιλαμβάνει συχνά πολλά μέρη κυρίως γιατί

1. τα φυσικά προβλήματα από μόνα τους αποτελούνται από πολλά επιμέρους υποπροβλήματα διαφορετικής φύσης

2. οι παράλληλες υπολογιστικές στρατηγικές και προσεγγίσεις απαιτούν πολλά (σχεδών ανεξάρτητα) επιμέρους προβλήματα, και

3. το υπάρχον λογισμό προσομοίωσης έχει εφαρμογή μόνο σε απλά γεωμετρικά σχήματα και μοντέλα φυσικής.

Ο κύριος σκοπός αυτής της διπλωματικής εργασίας είναι να προτείνει ένα περιβάλλον προσομοίωσης για την επίλυση των προβλημάτων πολλαπών χωρίων / πολλαπλών φυσικών προβλημάτων με τη μέθοδο της χαλάρωσης στις διεπαφές μεταξύ γειτονικών υποχωρίων του αρχικού προβλήματος *(Interface Relaxation).*

Το προτεινόμενο εργαλείο (*IRTool*) πρέπει να έχει αρκετές πολύ επιθυμητές ιδιότητες, όπως ευρεία εφαρμογή , αυξημ ένη προσαρμ οστικότητα, υψηλή απόδοση , παραλληλισμό και επαναχρησιμοποίηση λογισμικού, με την ενσωμάτωση μεθόδων σε διάφορους επιστημονικούς τομείς όπως η μαθηματική ανάλυση, αριθμητική ανάλυση, οι επιστημονικοί υπολογισμοί και οι κατανεμημένοι υπολογισμοί.

Abstract

The modeling and simulation of complex physical systems often involves many components because

1. the physical system itself has components of differing natures,
2. parallel computing strategies require many (somewhat independent) components, and
3. existing simulation software implies only to simpler geometrical shapes and physical situations.

The main purpose of this thesis is to propose a simulation environment for solving MultiDomain / MultiPhysics problems using different relaxation methods to the interfaces between neighboring subdomains of the original problem *(Interface Relaxation)*.

The proposed tool (IRTool) should have several very desirable properties as, fast convergence, wide application, increased flexibility, high performance, parallelism and reuse of existing software, integration of the advances in various scientific fields such as mathematical analysis, numerical analysis, scientific computing and distributed computing.

Table of Contents

List Of Acronyms

GEO	Geometric Construction Based Method
AVE	A simple method of averaging the solution and its normal derivative along the interfaces.
NEW	A scheme based on Newton’s method to “correct” the interface values.
ROB	An algorithm that uses Robin interface conditions for smoothing.
SCO	A scheme that is based (but not formulated) on Schur complement approach.
SHO	A method based on the concept of the shooting method for solving Ordinary Differential Equations (ODEs).
PDE	Partial Differential Equation
IR	Interface Relaxation
PDEToolbox	Partial Differential Equation Toolbox
IRToolbox	Interface Relaxation Toolbox

List of Figures

Chapter 1: Introduction

1.1 General

For the numerical solution of large partial differential equations (PDE’s) problems there are many techniques[1][2][11]. The first and most common approach is to discretize the geometrical domain using grids or meshes to create a large discrete problem. These grids or meshes are then partitioned to create a set of inter-connected discrete problems. This is simple *Domain Decomposition* (D.D. also known as sub-structuring) and the coupling between components (discrete problems) is rather tight as the mathematical model along interface points or elements is discretized into equations that involve details from both neighboring components. Moreover, domain decomposition methods solve a boundary value problem by splitting it into smaller boundary value problems on subdomains and iterating to coordinate the solution between adjacent subdomains. A coarse problem with one or few unknowns per subdomain is used to further coordinate solution between the subdomains globally. The problems on the subdomains are independent, which makes domain decomposition suitable for parallel computing. However domain decomposition methods are typically used as pre-conditioners, such as the conjugate gradient method.

The second and oldest approach is Schwarz Splitting which decomposes the geometrical domain into components with small overlap[3]. In overlapping domain decomposition methods, the subdomains overlap by more than the interface. The mathematical models on each component can then be solved independently in some way and the *Schwarz alternating method* is applied iteratively to compute the global solution. Of course, some discretization method is applied to the solution process on each individual component. The overlapping creates a serious complication in the Schwarz method even when the global problem has a simple geometry. The method has become more feasible with the discovery of non-overlapping domain versions.

The third and newest approach is *Interface Relaxation (IR)* where the geometrical domain is decomposed into subdomains, each with its own mathematical model. In these, non-overlapping methods, the subdomains intersect only on their interface. In primal methods, the continuity of the solution across subdomain interface is enforced by representing the value of the solution on all neighboring subdomains by the same unknown. Along the interfaces between subdomains one must satisfy interface conditions derived from the physical phenomena (e.g., continuity of mass or temperature, conservation of momentum). The models in each domain are solved in the inner loop of the interface relaxation iteration method to compute the global solution. The methods use one of a variety of “smoothing” formulas to reduce the error in satisfying the interface conditions. Finite element simulations of moderate size models require solving linear systems with millions of unknowns. Several hours per time step is an average sequential run time, therefore, parallel computing is a necessity.

The goals of handling different physical problems, using parallel computers and reusing existing software all lead to the need for high flexibility and loose coupling between components in the computation. The approaches mentioned above have similar goals but are quite different in their generality and flexibility. The tight coupling of domain decomposition requires that neighboring components have a lot shared information about their discretizations. Further, this approach is quite awkward when the models are different on neighboring components. The *mortar method* creates specialized refinements of the models and meshes along the interfaces to accommodate changes in models across interfaces. overlapping Schwarz methods are similarly constraint to a single physical problem for neighboring subdomains. the non - overlapping Schwarz methods are restricted to a single mathematical model for neighboring subdomains.

The interface relaxation approach imposes no coupling conditions, except those inherent in the mathematical models, and it provides maximum generality and flexibility. The interface relaxation method is the method that we are going to use in order to solve complex PDE problems. The IR methodology is an iterative procedure. First the initial problem is decomposed into smaller and simpler PDE subproblems, where there is no overlap between the neighboring subdomains. On all subdomain interface we use initial values, which we estimate. The next step is to solve each single PDE subproblem independently using the estimated values on the interfaces. If the solution that we compute is not the same as the real solution we improve the values on the interfaces using a relaxer. A relaxer is an interface relaxation method, such as GEO, ROB, AVE [12][13][14]. This procedure is progressed iteratively until satisfactory accuracy and convergence are achieved.

The main purpose of this dissertation is to propose a simulation environment for solving multi-domain/multi-physics problems. More specifically we want to solve elliptic PDE problems that are coupled with both cartesian and general decompositions. The Graphical User Interface (GUI) is a simulation framework where we can draw a 2-D complicated domain and define boundary and interface conditions. We can also specify the partial differential equation, create, inspect and refine the mesh and compute the solution for the particular problem. This framework must have several very desirable properties like fast convergence, wide applicability, increased adaptivity, high efficiency, inherent parallelism and software reuse by integrating advances in different scientific areas like mathematical analysis, numerical analysis, approximation, scientific computing, distributed computing and agent computing.

In the *Second Chapter*, we present previous related works. In the *Third Chapter* we provide a theoretical background about how *Interface Relaxation* can be used in order to solve complex PDE's problems. In the *Forth Chapter* a brief description of Matlab's PDETool is given. The *Fifth Chapter* introduces IRTool's Graphical User Interface. The PDE Toolbox is used as a base of the proposed simulating environment for the decomposition and solution with the use of various i*nterface relaxation methods*. In the Sixth Chapter we provide technical information for the implementation of our tool. In the Seventh Chapter a manual is given which provides any prospective user all the necessary guidance needed to compose the initial problem and afterwards decompose it and solve it with the Interface Relaxation Toolbox. The *Eighth Chapter* suggests guidelines for any future development and extension of the proposed implementation of the Interface Relaxation Toolbox.

Chapter 2: Related Works

2.1 First Implementation of IR

Given the fact that Methodology in Interface Relaxation is relatively new, it is easily assumable that there are only a few implementations that cover this area of research. A first but also naive of such a prototype implementation goes back to 1991. it was solely based on core TCP/IP routines to implement the collaboration among the co-workers. it did not use software parts technology but rather developed from scratch on local solvers and implemented just one relaxer. The second primitive implementation differs from the first one mainly on the fact that it exploited, based on plain KQML messages, the Agent approach to integrate the ELLPACK PSE. both were very unstable, were used through text based user interfaces, did not complied with standard technology and were very limited for our purposes [5][6][7][8][9]. Nevertheless, the later implementation have provided with a good starting point and it is presented next.

2.2 SciAgent

Collaborative PDE Solvers: Theory and Practice[2], presented in 2000 an implementation of a collaborative PDE system, named SciAgent, for truly heterogeneous distributed computer systems. In particular the architecture and the main software components are described and the Agent Technology used is introduced. SciAgent implementation consists of a whole class of Interface Relaxation methods in an agent based framework that is implemented mainly using C and JAVA. This implementation is used for general two-dimensional decompositions of linear and non-linear elliptic PDE problems. The SciAgents exploit the inherent parallelism in the interface relaxation methods using the Agents computing paradigm over a network of heterogenous workstations. The components of a SciAgent system are shown in Figure 2.1 below.

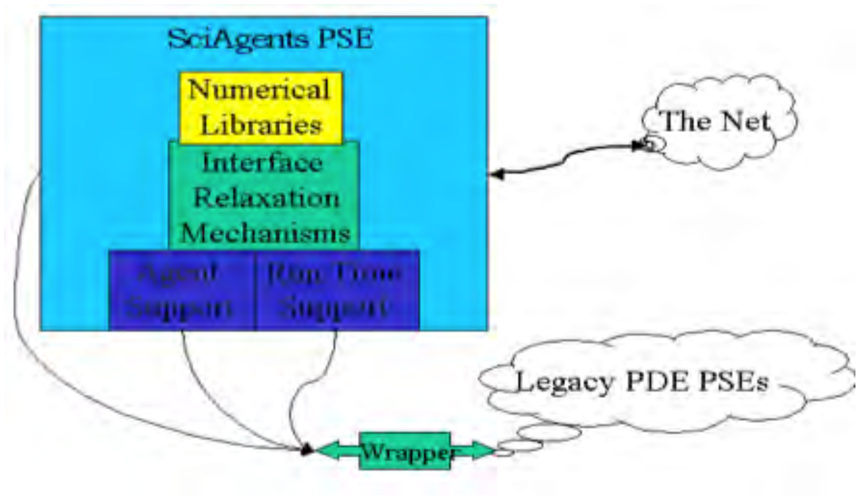


Figure 2.1: The components of the SciAgent System

Specifically, the SciAgents transform the physical problem into a network of local PDE solvers and interface relaxers. In the SciAgent prototype there are three types of agents: one PDECoordinator agent, several PDESolver and PDEMediator agents. The PDECoordinator, as it is Figured by its name, is responsible for the control of the entire application and coordinate the whole procedure, a PDEMediator arbitrates between the two solvers sharing a boundary between two domains, and a PDESolver is a wrapper for the legacy application and solves the local problem.

When the PDECoordinator agent is started, reads a problem description file and writes the information into its model. The input file contains information about the number of solvers and mediators, the characteristics of the interfaces, the initial guesses on the interfaces, the interface relaxation methods and the names of the machines that will be used to solve the whole problem. The next step in the procedure is to create and conFigure the PDESolver and PDEMediator agents. The PDECoordinator uses their addresses to setup the communication between them. Then the coordinator waits for messages from the mediators, regarding the status of the convergence to the solution of the problem, or from the user. The messages from the user are to change the values of specific variables of the input file, such as convergence tolerance or to force the execution to stop.

The PDESolver agent is responsible to solve the problem locally and path the input/ output files etc. In the first state, the PDESolver starts-up the Pelltool which compiles the .e file that describes the local PDE problem, and creates the executable that will be used later on by the ExecuteTool. Both Pelltool and ExecuteTool are parts of PELLPACK system. In the next state, the solver extracts the points on the interfaces from the file that contains the mesh/ grid points, and writes them into a file. Then the solver notifies the mediators that the files are ready. The PDESolver agent remains idle until being notified by the mediator that the list with all the points and their initial guesses are stored in a file at a specific location. Then the solver uses these files to run the ExecuteTool to solve the problem and then, the solver send a message to the mediators that new values are computed, and waits for their response. Depending on the message from the mediators, the solver will solve the problem again, remain idle waiting for the other solvers, or plot the local solution. The PDECoordinator is able to terminate the PDESolver by sending an appropriate message.

The mediator agent, PDEMediator, is created and conFigured by the PDECo- ordinator agent. The mediator agent has a complete description of the interface, the relaxation method used, the solvers to collaborate with, the location of the input/ output files, the location of the legacy programs, the tolerance used to decide convergence, and the initial guess function. This information is provided by the coordinator agent. After being started, the mediator waits for the boundary points from the neighboring solvers. In the next stage, the mediator combines the two point lists and then uses the initial guess to compute values at these points. Afterwards, the mediator sends a message to the two solvers that the files with the points and their values are ready. The mediator remains idle, waiting for new values from the two solvers. When it receives new values it moves to the next stage, reads the new data and compares them with current data. Then the mediator agent uses the relaxation method to calculate the new values for the boundary conditions. If convergence is reached on this interface then the mediator sends messages to the solvers and informs the PDECoordinator about the local convergence so it will be able to decide on global convergence. A message from the coordinator is sent to the mediator and the PDEMediator will finish, in case of global convergence or wait for new data from the two solvers.

SciAgents require strong interpolation support, procedure for estimating initial guesses, mechanisms for determining “good” values for relaxation parameters and criteria to control the iterative procedure. Interpolation is needed because grids/meshes do not necessarily match on the interface segment. Also the estimation of initial values on the interface is a very complex procedure, because the Neumman and the mixed boundary conditions are sensitive to their initial guesses and as the PDE problems get more complicated better initial guesses will be needed.

Chapter 3: Theoretical Background

3.1 General

The various domain decomposition methods that have been recently developed for the efficient solution of elliptical differential equations can be easily classified into two categories - overlapping and non-overlapping [3]. Both approaches already have been used to effectively model large scale, industrial, ill-conditioned problems. Nevertheless it is believed that further theoretical and experimental analysis is required before such methods will become practical and useful tools for non - experts.

Overlapping (Schwarz) schemes have received in the past a great deal of attention. Articles that review and compare various such schemes and survey the associated preconditioning strategies have already appeared in the literature. it is relatively recent that a number of studies have shown that non-overlapping schemes can compete well an d can possibly free the user from certain complications in their formulation and implementation. the comparison of the main characteristics of these two classes of methods and the existence of equivalent relations between them have already received a great deal of study.

Interface relaxation methods takes us a step beyond non overlapping domain decomposition. In an effort to mimic the physics in the real world, they split a complicated partial differential equation (PDE) that acts on a large and / or complex domain into a set of PDE problems with different but simple operators acting on different and easy to solve subdomains. This Multi - PDE, Multi - Domain system is properly coupled using smoothing operations on the inter domain boundaries.

From the interface relaxation viewpoint, the methods that can be used in order to solve a problem, consist of partitioning the domain on a set of non-overlapping subdomains and of imposing some boundary conditions on the interface boundaries defined by this partition. Then, using initial guesses on the interfaces, the set of the resulting PDE problems is solved. The solutions obtained do not satisfy the interface boundary conditions and interface relaxation is applied to obtain new interface boundary values, which satisfy the conditions better, and we solve the PDEs with these new values. We repeat the above steps until the desired convergence is acquired.

For this dissertation we have collected most of the known interface relaxation methods and for the implementation of the IRTool we have used the GEO method.
Specifically, we consider the methods listed below in alphabetical order with respect to their acronyms. These acronyms are used in the sequel to refer to associated methods.

AVE A simple method of averaging the solution and its normal derivative along the interfaces.

GEO A method based on a simple geometric contraction.

NEW A scheme based on Newton’s method to “correct” the interface values.

ROB An algorithm that uses Robin interface conditions for smoothing.

SCO A scheme that is based (but not formulated) on Schur complement approach.

SHO A method based on the concept of the shooting method for solving Ordinary Differential Equations (ODEs).

SPO A method originated from the use of Steklov Poincare operator which involves alternating boundary condition types.

3.2 Domain Decomposition & Interface Relaxation

The domain decomposition world consists of two parts, overlapping and non-overlapping. Overlapping, known also as Schwarz, methods were the first considered and have already proved themselves as very efficient numerical procedures enjoying certain very desirable convergence properties. Nevertheless, it has been also observed that they might have serious drawbacks which will prohibit their use for certain applications. For example, almost all of the many proposed domain decomposition methods for solving wave propagation models (that consist of the Helmoltz equation coupled with various absorbing or reflecting boundary conditions) are non-overlapping.

Non overlapping methods exhibit certain advantages compared to overlapping ones. Specifically:

- They are not sensitive to jumps on the operator coefficients. Their convergence behavior and theoretical errors estimates remain the same even if the differential operator includes discontinuous coefficients provided that the jumps occur along the interface lines.

- They have smaller communication overhead in a parallel implementation on distributed memory multiprocessor systems. Their communication overhead is proportional to the length of the interface lines while it is proportional to the overlapping are in the case of overlapping methods.

- The bookkeeping is rather easy for the decomposition and manipulations of the associated data structures compared to the more complicated and costly bookkeeping of the overlapping methods.

Interface relaxation is a step beyond non overlapping domain decomposition; it follows Southwell’s relaxation of the 1930’ but at the PDE instead of the linear algebra level - to formulate relaxation as iterated interface smoothing procedures. A complex physical phenomenon consists of a collection of simple parts with each one of them obeying a single physical law locally and adjusting its interface conditions with neighbors. Interface relaxation partitions the domain on a set of non overlapping subdomains, imposes some boundary conditions on the interface among subdomain lines. given an initial guess, it imitates the physics of the real world by solving the local problems exactly on each subdomain and relaxing boundary values to get better estimates of correct interface conditions. The procedure described above can be explained as an algorithm, as follows :

1. *Guess solution values (and derivatives if needed) on all subdomain interfaces.*

2. *Solve all single PDEs exactly and independently on all the subdomain with these values as boundary conditions.*

3. *Compare and improve the values on all interfaces using a relaxer*

4. *Return to Step 2 until satisfactory accuracy is achieved.*

The simplest relaxers do some sort of “smoothing” of values on the interfaces and averaging is a good mental model for a relaxation formula.

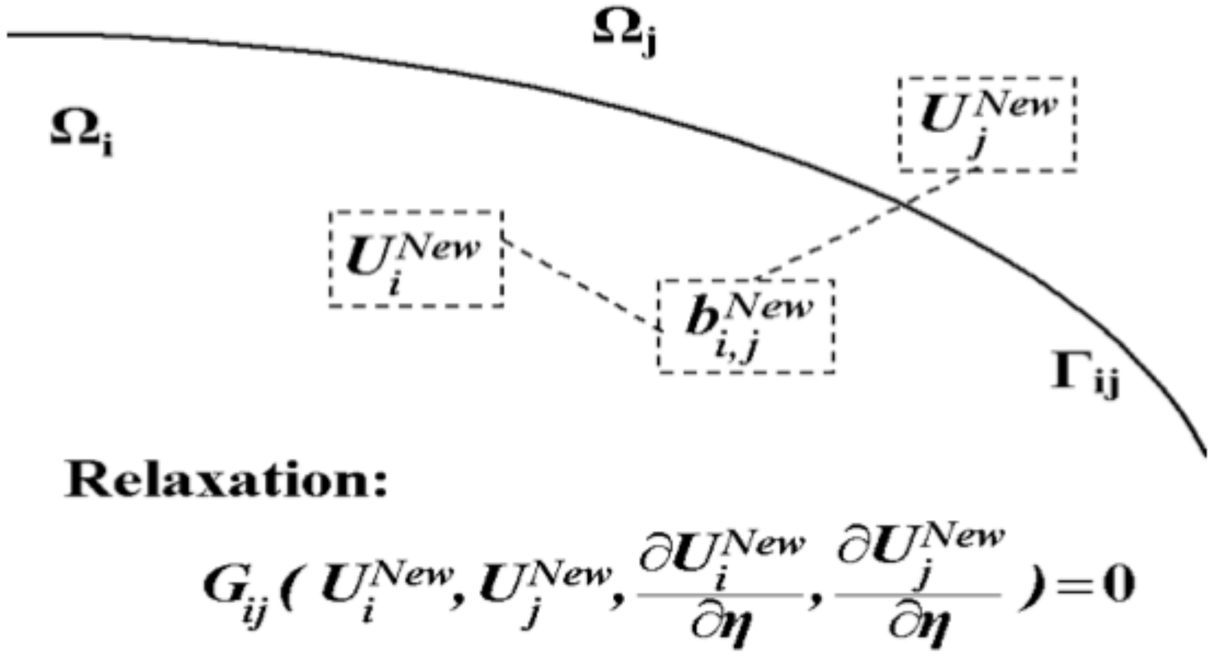

Ω_j
Ω_i
U_j^{New}
U_i^{New}
$b_{i,j}^{New}$
Γ_{ij}
Relaxation:
$G_{ij}(U_i^{New}, U_j^{New}, \frac{\partial U_i^{New}}{\partial \eta}, \frac{\partial U_j^{New}}{\partial \eta}) = 0$

Figure 3.2.1: The Interface Relaxation mechanism

The illustrated Figure above shows the generic relaxation formula G_{ij} (based on the current solutions U_i^{new} and U_j^{new} of the two local to the neighboring subdomains Ω_i and Ω_j) calculates successive approximations $b_{i,j}^{New}$ to the solution on the interface $\Gamma_{i,j}$ between them.

To formally describe this method we consider the differential problem

$$Du = f \ in \ \Omega, \qquad Bu = c \ on \ \partial\Omega \tag{3.1}$$

where D is an elliptic, non-linear in general, differential operator and B a condition operator defined on the boundary $\delta\Omega$ of an open domain Ω E R^d , d=1,2,... This domain is partitioned into p open subdomains Ω_i ,i=1,...,p such that

$$\Omega = \cup_{i=1}^{p} \overline{\Omega_i} \backslash \partial\Omega \text{ and } \cap_{i=1}^{p} \Omega_i = \emptyset. \tag{3.2}$$

For reasons related either to the physical characteristics of this problem or to the computing resources available, one would like to replace (3.1) with following system of loosely coupled differential problems

$$\begin{gathered} D_i u = f_i \ in \ \Omega_i, \\ G_{ij} u = 0 \ on \ (\partial\Omega_i \cap \partial\Omega_j) \backslash \partial\Omega \ \ \forall j \neq i, \qquad B_i u = c_i \ on \ \partial\Omega_i \cap \partial\Omega \end{gathered} \tag{3.3}$$

where i=1,...,p. These differential problems are coupled through the interface conditions $G_{ij}u=0$ and involve the restrictions D_i and B_i of the global differential and boundary operators, D and B, respectively, on each subdomain with some of them possibly linear and some other nonlinear. The functions f_i and c_i are similar restrictions of the function f and c. The local interface operator G_{ij} is associated with the interface relaxation method and different selections for the G_{ij}'s lead to different relaxation schemes. In this dissertation we consider interface relaxation methods that have the following characteristics.

- They first decompose the problem (3.1) at differential level and then discretize the resulting differential subproblems (3.3).
- They have the versatility to use the most appropriate discretization scheme for each subproblem.
- They do not overlap the subdomains Ω_i
- using good relaxation parameters in G_i , they are fast enough so no preconditioning is needed.
- They simplify the geometry and physics of the computation by considering the subproblems (3.3) instead of the global differential problem (3.1).
- They can utilize software parts technology by reusing existing “legacy” software parts for solving the individual subproblems (3.3).
- They are general and robust.

3.3 Interface Relaxation Methods

Due to the inherent abstraction, it is relatively easy to describe the various interface smoothing methods at both the conceptual and algorithmic level. For the purpose of this dissertation though, we will present only the GEO method which is used for the implementation of our tool. For simplicity in the presentation of algorithms, we consider only one way (along the x axis) partition of the domain. Therefore each subdomain can have at maximum two interface lines with the two neighbouring subdomains. The basic building block for our algorithm is the procedure u = solve_pde(ui,dui) which calculates the solution u of the local to a subdomain PDE problem with Dirichlet, Neumann or GEO boundary conditions on the interface using as the interface values ui and its gradient dui. The subscripts R and L denote left and right subdomains or interfaces respectively and u_i denotes the solution of the problem associated with subdomain Ω_i.

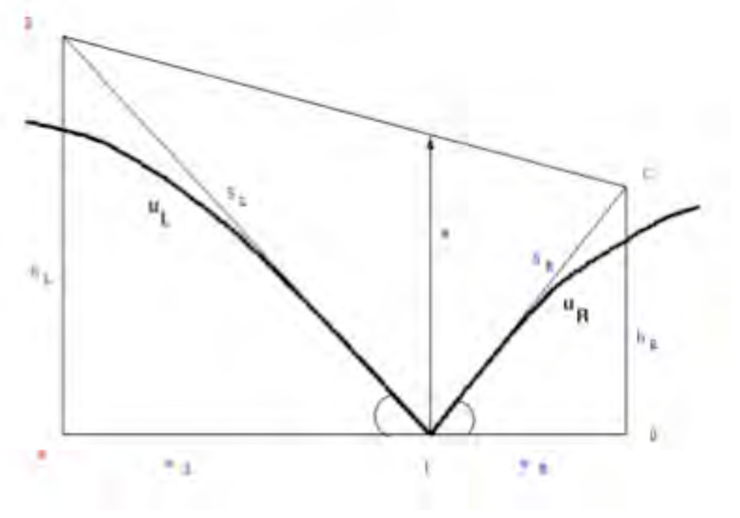


Figure 3.3.1: Cross section perpendicular to the interface where U_L and U_R have slopes S_L and S_R at the interface point I. Changing the values of U_L and U_R by a quantity m makes the slopes equal in magnitude.

3.3.1 The Geometric (GEO) Construction Based Method

GEO estimates the new solution for each subdomain by solving a Dirichlet problem and is classified as an *one step* method. The values on the interfaces are obtained by adding to the old ones, a geometrical weighted combination of the normal boundary derivatives of the adjacent subdomains. Specifically, we assume in Figure 3.3.1 that U_L and U_R are the solutions of the PDE problems associated with the left and right subdomains, respectively, of the interface point *I*.
They are equal along interface *I* and we denote by S_L and S_R their slopes at *I*. As it can be easily seen geometrically, *m* is the correction needed to be added to U_L and U_R so as to match the normal derivatives *I*. To calculate *m* we consider the two triangles *IAB* and *CDI* whose heights are given by multiplying the corresponding tangent with the base of the triangle, or equivalently by multiplying the normal derivative with the base. The bases w_L and w_R are the widths assumed for the validity of the slope values; these can be arbitrary selected and play the role of the relaxation parameters. The new interface values are now given by adding the weighted average of the heights to the old interface values U_L and U_R. One can intuitively view this as grabbing the function *U* at the *I* and stretching it up by *m* until its derivative becomes continuous. numerical experiments show that the convergence rate does not seem to depend much on the widths w_L and w_R. In case that U_L and U_R are not equal on *I* we simply use their average. ***GEO*** is given algorithmically by

```
• for k = 0,1,2, ...
    – $ui^{(k+1)} = \frac{u_L^{(k)}+u_R^{(k)}}{2} - \frac{w_L w_R}{w_L+w_R}\left(\frac{\partial u_L^{(k)}}{\partial x} - \frac{\partial u_R^{(k)}}{\partial x}\right)$ on each interface
    – $u^{(k+1)} = \text{solve\_pde}(ui^{(k+1)})$ in each subdomain
```

Chapter 4: System Design & PDETool

4.1 Matlab PDETool

4.1.1 What does this Toolbox do?

The *Partial Differential Equation* (PDE) Toolbox [4] provides a powerful and flexible environment for the study and solution of partial differential equations in two space dimensions and time. The equations are discretized by the Finite Element Method (FEM). The objectives of the PDE Toolbox are to provide you with tools that:

- Define a PDE problem, i.e., define 2-D regions, boundary conditions, and PDE coefficients.
- Numerically solve the PDE problem, i.e., generate unstructured meshes, discretize the equations, and produce an approximation to the solution.
- Visualize the results.

4.1.2 Who can use this toolbox?

The PDE Toolbox is designed for both beginners and advanced users. The minimal requirement is that you can formulate a PDE problem on paper (draw the domain, write the boundary conditions, and the PDE). Start MATLAB. At the MATLAB command line type:

pdetool

This invokes the graphical user interface (GUI), which is a self-contained graphical environment for PDE solving. For common applications you can use the specific physical terms rather than abstract coefficients. Using pdetool requires no knowledge of the mathematics behind the PDE, the numerical schemes, or MATLAB.

Advanced applications are also possible by downloading the domain geometry, boundary conditions, and mesh description to the MATLAB workspace. From the command line (or M-files) you can call functions from the toolbox to do the hard work, e.g., generate meshes, discretize your problem, perform interpolation, plot data on unstructured grids, etc., while you retain full control over the global numerical algorithm.

4.1.3 What Problems can I solve?

The basic equation of the PDE Toolbox is the PDE

$$-\nabla\cdot(c\nabla u)+au=f \text{ in } \Omega,$$

which we shall refer to as the *elliptic equation*, regardless of whether its coefficients and boundary conditions make the PDE problem elliptic in the mathematical sense. Analogously, we shall use the terms *parabolic equation* and *hyperbolic equation* for equations with spatial operators like the one above, and first and second order time derivatives, respectively. Ω is a bounded domain in the plane. *c*, *a*, *f*, and the unknown *u* are scalar, complex valued functions defined on Ω. *c* can be a 2-by-2 matrix function on Ω. The toolbox can also handle the parabolic PDE

$$d\frac{\partial u}{\partial t}-\nabla\cdot(c\nabla u)+au=f,$$

the hyperbolic PDE

$$d\frac{\partial^2 u}{\partial t^2} - \nabla \cdot (c\nabla u) + au = f,$$

and the eigen value problem

$$-\nabla \cdot (c\nabla u) + au = \lambda du$$

where d is a complex valued function on Ω, and λ is an unknown eigenvalue. For the parabolic and hyperbolic PDE the coefficients c, a, f, and d can depend on time. A nonlinear solver is available for the nonlinear elliptic PDE

$$-\nabla \cdot (c(u)\nabla u) + a(u)u = f(u) \,,$$

where c, a, and f are functions of the unknown solution u. All solvers can handle the system case

$$-\nabla \cdot (c_{11}\nabla u_1) - \nabla \cdot (c_{12}\nabla u_2) + a_{11}u_1 + a_{12}u_2 = f_1$$
$$-\nabla\cdot(c_{21}\nabla u_1)-\nabla\cdot(c_{22}\nabla u_2)+a_{21}u_1+a_{22}u_2 = f_2 \,.$$

You can work with systems of arbitrary dimension from the command line. For the elliptic problem, an adaptive mesh refinement algorithm is implemented. It can also be used in conjunction with the nonlinear solver. In addition, a fast solver for Poisson’s equation on a rectangular grid is available.The following boundary conditions are defined for scalar u:

- *Dirichlet*: $hu = r$ on the boundary $\partial\Omega$.
- *Generalized Neumann*: $n \cdot (c\nabla u) + qu = g$ on $\partial\Omega$.

n is the outward unit normal. g, q, h, and r are complex valued functions defined on $\partial\Omega$. (The eigenvalue problem is a homogeneous problem, i.e., g = 0, r = 0.) In the nonlinear case, the coefficients, g, q, h, and r can depend on u, and for the hyperbolic and parabolic PDE, the coefficients can depend on time. For the two-dimensional system case, Dirichlet boundary condition is

$$h_{11}u_1 + h_{12}u_2 = r_1$$
$$h_{21}u_1 + h_{22}u_2 = r_2 \,,$$

the generalized Neumann boundary condition is

$$n\cdot(c_{11}\nabla u_1)+n\cdot(c_{12}\nabla u_2)+q_{11}u_1+q_{12}u_2 = g_1$$
$$n\cdot(c_{21}\nabla u_1)+n\cdot(c_{22}\nabla u_2)+q_{21}u_1+q_{22}u_2 = g_2,$$

and the *mixed* boundary condition is

$$h_{11}u_1 + h_{12}u_2 = r_1$$
$$n\cdot(c_{11}\nabla u_1)+n\cdot(c_{12}\nabla u_2)+q_{11}u_1+q_{12}u_2 = g_1+h_{11}\mu$$
$$n\cdot(c_{21}\nabla u_1)+n\cdot(c_{22}\nabla u_2)+q_{21}u_1+q_{22}u_2 = g_2+h_{12}\mu,$$

where µ is computed such that the Dirichlet boundary condition is satisfied. Dirichlet boundary conditions are also called *essential* boundary conditions, and Neumann boundary conditions are also called *natural* boundary conditions.

4.1.4 In which areas can the Toolbox be used?

The PDEs implemented in the toolbox are used as a mathematical model for a wide variety of phenomena in all branches of engineering and science. The following is by no means a complete list of examples:
The elliptic and parabolic equations are used for modeling

- steady and unsteady heat transfer in solids
- flows in porous media and diffusion problems
- electrostatics of dielectric and conductive media potential flow

The hyperbolic equation is used for

- transient and harmonic wave propagation in acoustics and electromagnetic
- transverse motions of membranes

The eigenvalue problems are used for, e.g.,

- determining natural vibration states in membranes and structural mechanics problems

Last, but not least, the toolbox can be used for educational purposes as a complement to understanding the theory of the Finite Element Method.

4.1.5 How do I define a PDE Problem?

The simplest way to define a PDE problem is using the graphical user interface (GUI), implemented in pdetool. There are three modes that correspond to different stages of defining a PDE problem:

- Draw mode, you create Ω, the geometry, using the constructive solid geometry (CSG) model paradigm. A set of solid objects (rectangle, circle, ellipse, and polygon) is provided. You can combine these objects using *set formulas*.
- In Boundary mode, you specify the boundary conditions. You can have different types of boundary conditions on different boundary segments.
- In PDE mode, you interactively specify the type of PDE and the coefficients *c*, *a*, *f*, and *d*. You can specify the coefficients for each subdomain independently. This may ease the specification of, e.g., various material properties in a PDE model.

4.1.6 How can I solve a PDE Problem?

Most problems can be solved from the graphical user interface. There are two major modes that help you solve a problem:

- In Mesh mode, you generate and plot meshes. You can control the parameters of the automated mesh generator.

- In Solve mode, you can invoke and control the nonlinear and adaptive solvers for elliptic problems. For parabolic and hyperbolic problems, you can specify the initial values, and the times for which the output should be generated. For the eigenvalue solver, you can specify the interval in which to search for eigenvalues.

After solving a problem, you can return to the Mesh mode to further refine your mesh and then solve again. You can also employ the adaptive mesh refiner and solver. This option tries to find a mesh that fits the solution.

4.1.7 Can I use the Toolbox for Nonstandard problems?

For advanced, nonstandard applications you can transfer the description of domains, boundary conditions etc. to your MATLAB workspace. From there you use the functions of the PDE Toolbox for managing data on unstructured meshes. You have full access to the mesh generators, FEM discretizations of the PDE and boundary conditions, interpolation functions, etc. You can design your own solvers or use FEM to solve subproblems of more complex algorithms.

4.1.8 Who can I visualize my results?

From the graphical user interface you can use Plot mode, where you have a wide range of visualization possibilities. You can visualize both inside the pdetool GUI and in separate Figures. You can plot three different solution properties at the same time, using color, height, and vector field plots. Surface, mesh, contour, and arrow (quiver) plots are available. For surface plots, you can choose between interpolated and flat rendering schemes. The mesh may be hidden or exposed in all plot types. For parabolic and hyperbolic equations, you can even produce an animated movie of the solution’s time-dependence. All visualization functions are also accessible from the command line.

4.1.9 Are there any applications already implemented?

The PDE Toolbox is easy to use in the most common areas due to the application interfaces. Eight application interfaces are available, in addition to the generic scalar and system (vector valued *u*) cases:

- Structural Mechanics - Plane Stress
- Structural Mechanics - Plane Strain
- Electrostatics
- Magnetostatics
- AC Power Electromagnetics
- Conductive Media DC
- Heat Transfer
- Diffusion

These interfaces have dialog boxes where the PDE coefficients, boundary conditions, and solution are explained in terms of physical entities. The application interfaces enable you to enter specific parameters, such as Young's modulus in the structural mechanics problems. Also, visualization of the relevant physical variables is provided.
Several nontrivial examples are included in this manual. Many examples are solved both by using the GUI and in command-line mode.
The toolbox contains a number of demonstration M-files. They illustrate some ways in which you can write your own applications.

4.2 IRToolbox (Interface Relaxation Toolbox)

4.2.1 Expanding Matlab's PDE Toolbox

The PDE Toolbox is written using MATLAB's open system philosophy. There are no black-box functions, although some functions may not be easy to understand at first glance. The data structures and formats are documented. Anyone can examine the existing functions and create his / her own as needed.

4.2.2 Architecture's Basic Idea & Functionality

The IRTool (Interface Relaxation Tool) is a Matlab Toolbox that solves MultiDomain MultiPhysics Problems. Firstly we studied PDETool which as mentioned before solves 2 - Dimensional PDE problems. Although PDETool can face multi-domain problems at the linear algebra level of study (domain decompositions techniques), it is not able to solve multiDomain multiPhysics problems with Interface Relaxation (IR) Methods. The common boundary between different domains of a PDE problem, named interfaces, can be treated as the physics demand in a high level of decomposition by the IR methodology. We embedded PDEtool in our toolbox, IRTool, in order to be able to define multiPhysics / multiDomain PDEs, set appropriate conditions on the global boundary, define/call proper methods on the interfaces, solve the global problem and visualize the solution of the entire PDE problem. Our work was quite difficult since we had to understand the MATLAB code, find out the proper data structures the MATLAB uses to store and compute a PDE problem and then use them properly in our toolbox to define and solve a composed PDE problem by making multiple PDETools communicating properly, which in PDETool case it does not allow us to create more than one design window.

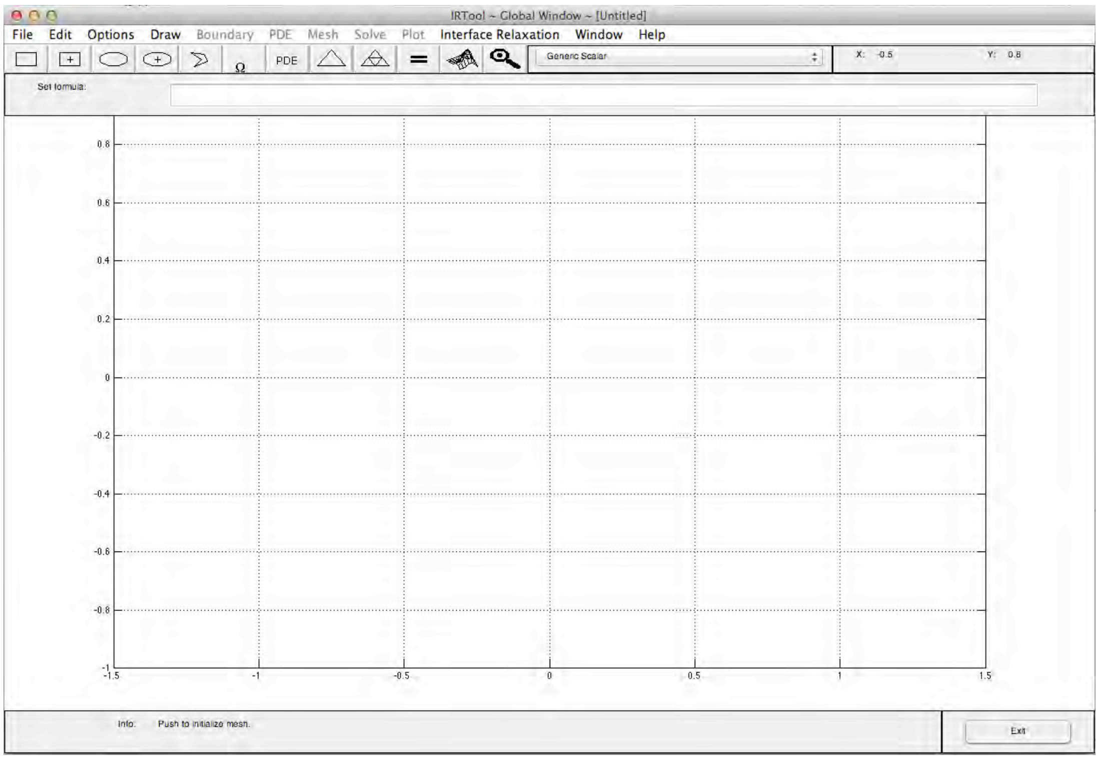
Chapter 5: IRToolbox GUI

5.1 Global Window

The PDE Toolbox which is implemented in Matlab provides a flexible environment for the study and solution of partial differential equations. Nevertheless, it does not handle with interface relaxation methods the common boundaries of a multi domain problem. This is the reason that inducts us to implement the IR Toolbox. The Interface Relaxation Toolbox provides a graphical user environment for the study and solution of partial differential equations with multiple domains / multiple physics as it considers at the same time the interface relaxation methodology. Therefore, the IRToolbox has some additional properties and capabilities. The basic need was that with the IRTool we can deal with the interface segments and solve the global problem by contemplating parameters such as interface relaxation method, initial guesses on the interface segments and tolerance for the convergence of the global solution.

The Graphical User Interface (GUI) of IRTool is an extension of PDETool's GUI with some modifications that are implied by the IR methodology. IR has a pull-down menu bar that we can use to control the modeling. It conforms to common pull-down menu standards. Menu items followed by a right arrow lead to a submenu. Menu items followed by an ellipsis lead to a dialog box. Stand-alone menu items lead to direct action. Some menu items can be executed by using keyboard accelerators. IRTool also contains a toolbar with icon buttons for quick and easy access to some of the most important functions. The following sections describe the contents of IRTool menus, the dialog boxes associated with menu items and an illustrated example.

To get started with the graphical environment we simply type in the Matlab prompt the command ***irtool***. The GUI looks like the Figure below (Figure 5.1) and is the initial / global window of our implementation.



Info: Push to initialize mesh.

Figure 5.1: IRTool Global Window

5.1.1 File Menu

The pull-down menu has the following applications: New, Open, Save, Save As, Print, Exit. These applications deal with basic procedures that exist in many toolboxes of Matlab.

5.1.2 Edit Menu

The Edit menu provides the applications below: Undo, Cut, Copy, Paste, Clear, Select All.

5.1.3 Options Menu

Similarly, the Options menu is the same as the pull-down Options menu of the pdetool. Its applications are: Grid, Grid spacing, Snap, Axis Limits, Axis Equal, Turn off Toolbar Help, Zoom, Application and Refresh.

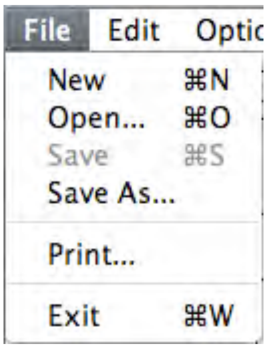


Figure 5.1.1: IRTool File Menu

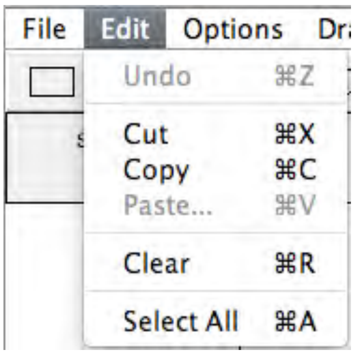


Figure 5.1.2: IRTool Edit Menu

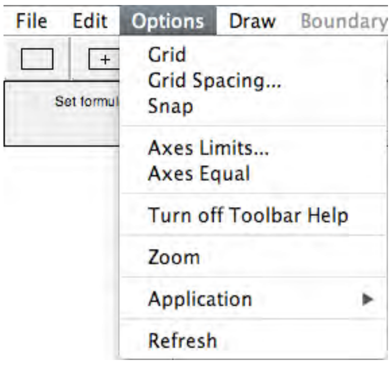


Figure 5.1.3: IRTool Options Menu

5.1.4 Draw Menu

In the Draw Mode, we can draw the geometry on which we want to solve the PDE. As in PDE Toolbox, the IRTool provides four types of solid objects: polygons, rectangles, circles, and ellipses by the selecting the following options from the pulldown menu of the Draw menu: Rectangle/Square, Rectangle/square (centered), Ellipse/Circle, Ellipse/Circle (centered), Polygon. In order to rotate the selected objects we can use the Rotate choice from the same menu.

The objects are used to create a Constructive Solid Geometry model (CSG model). Each solid object is assigned a unique label, and by the use of set algebra, the resulting geometry can be made up of a combination of unions, intersections, and set differences. By default, the resulting CSG model is the union of all solid objects. The only thing that we must take care when we draw the geometry is that the solid objects must not overlap each other if we want to solve the PDE problem using the interface relaxation methodology.

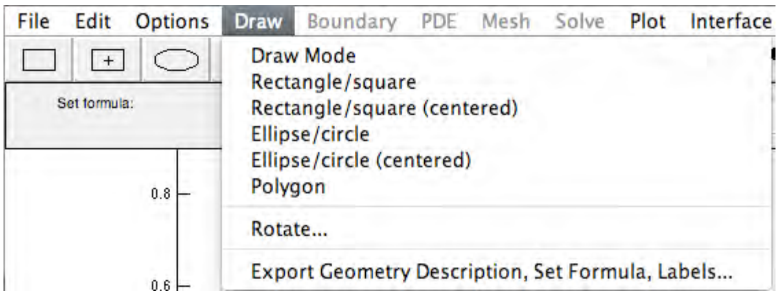


Figure 5.1.4: IRTool Draw Menu

5.1.5 Interface Relaxation Menu

Interface Relaxation Menu in the *global window* provides us with the following two choices

A. *Decompose Geometry*

Splits the initial geometry to the consisting subdomains. Each subdomain is redrawn to individual window - *subdomain window* - awaiting for further specification, i.e. boundary conditions, interface relaxation conditions e.t.c.

B. *Parameters*

Opens additional window for user entry. The user has to specify the number of iterations that the interface method will execute and the error tolerance (convergence). The window by default has the iteration number set to 20 and the error value set to 0.5e-6.

C. *Solve MultiPDE*

Starts the solution of all the subdomains & interfaces.

D. *Clear IR Parameters*

In case that we want to insert new boundary values,pde coefficients e.t.c. we have first to choose this menu in order the old parameters to be flashed out from our subdomains.

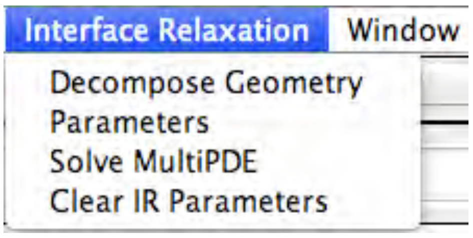
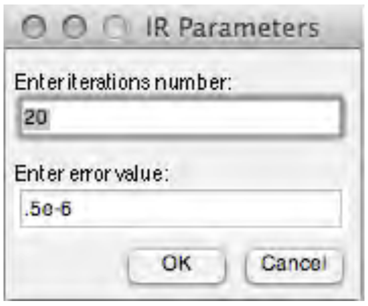


Figure 5.1.5a: IRTool Interface Relaxation Menu



Enter iterations number: 20

Enter error value: .5e-6

Figure 5.1.5b: IR Parameters pop - up window

5.2 Subdomain Window

Supposing that we have the initial / global geometry shown in Figure 5.2a, pressing the Decompose geometry from the Interface Relaxation menu, the current geometry splits in multiple windows -*subdomain windows*-. Every window now creates a unique *handle* for every subdomain as shown in Figures 5.2b and 5.2c respectively.

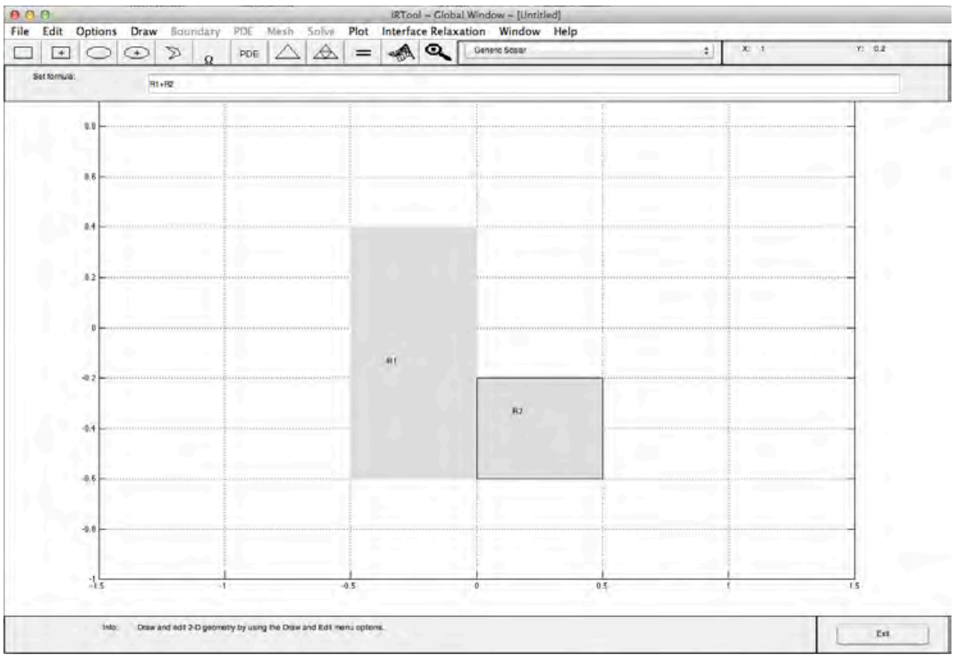
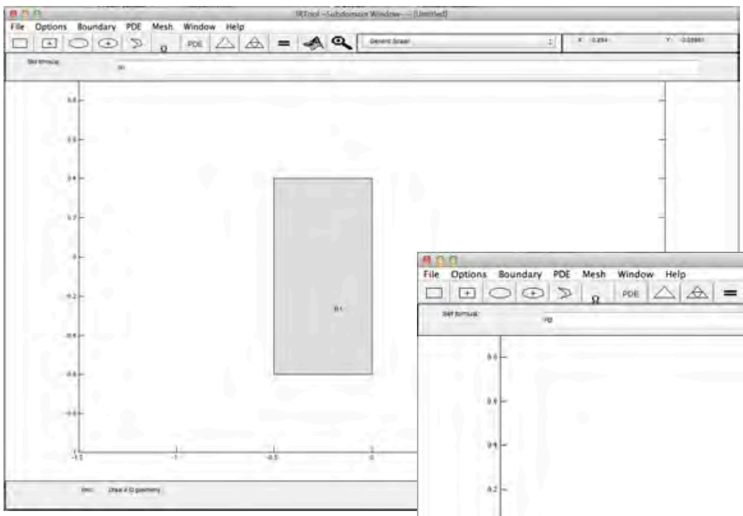

IRTool – Global Window – [Untitled]
File Edit Options Draw Boundary PDE Mesh Solve Plot Interface Relaxation Window Help
Generic Scalar
X: 1 Y: 0.2
Set formula: R1+R2
R1
R2
Info: Draw and edit 2-D geometry by using the Draw and Edit menu options.
Exit

Figure 5.2a: Global Window with drawn Geometry


IRTool –Subdomain Window– [Untitled]
File Options Boundary PDE Mesh Window Help
Set formula: R1
R1
Info: Draw 2-D geometry

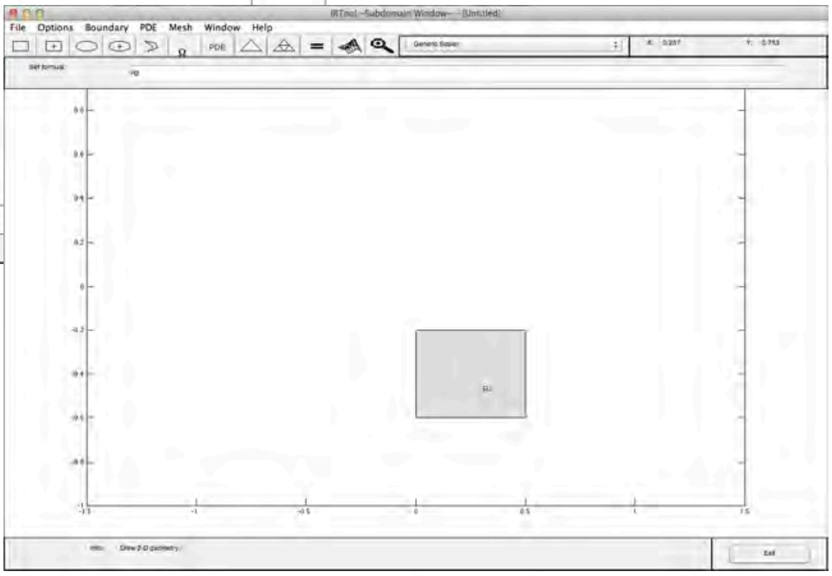

IRTool –Subdomain Window– [Untitled]
File Options Boundary PDE Mesh Window Help
Set formula: R2
R2
Info: Draw 2-D geometry
Exit

Figure 5.2b: Subdomain Window for the first domain

Figure 5.2c: Subdomain Window for the second domain

5.2.1 Boundary Menu

In the Boundary menu we have the following options:Boundary mode, Specify Boundary Conditions, Show Edge Labels, Show Sub-domains Labels, Remove Sub-domain Border, Remove All Sub-domain Borders and Export Decomposed Geometry, Boundary Cond’s, as shown in Figure 5.2.1a.

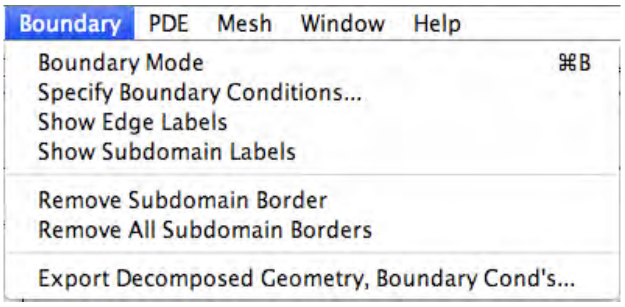
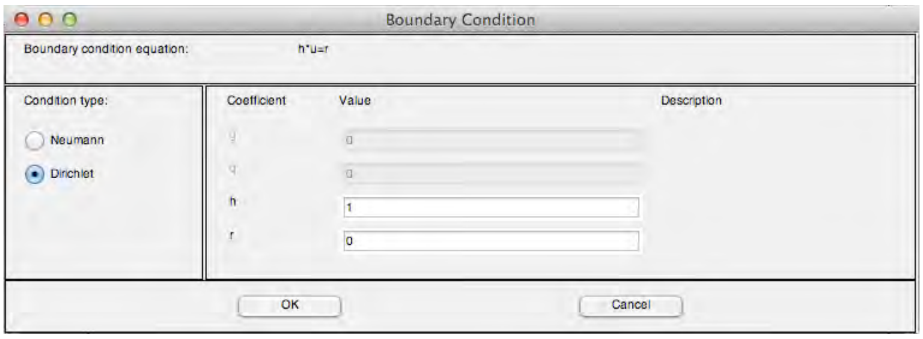

Boundary
PDE
Mesh
Window
Help
Boundary Mode ⌘B
Specify Boundary Conditions...
Show Edge Labels
Show Subdomain Labels
Remove Subdomain Border
Remove All Subdomain Borders
Export Decomposed Geometry, Boundary Cond's...

Figure 5.2.1a: Boundary Menu

We can now define the boundary conditions for the outer boundaries but for the interfaces as well. We can enter the Boundary Mode by clicking the ∂Ω icon, or by selecting Boundary Mode from the Boundary menu. The boundaries are indicated by colored lines with arrows. The boundary condition can also be a function of x and y, or simply a constant. By default, the boundary condition is of Dirichlet type: u = 0 on the boundary. If we double click on a boundary segment the boundary condition window pops up as shown in Figure 5.2.1b


Boundary Condition

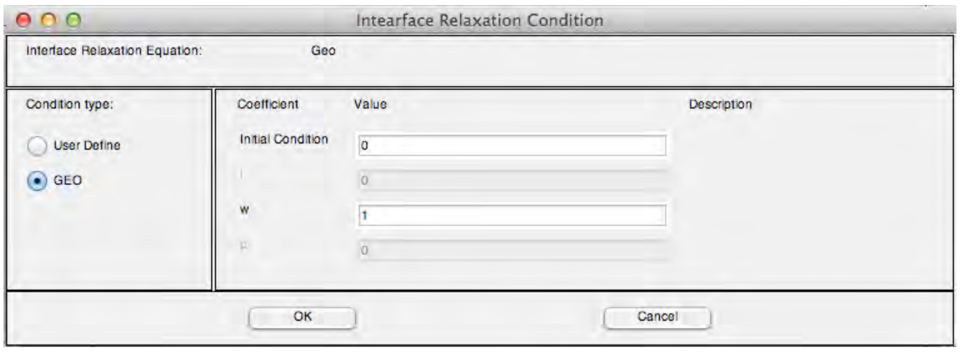
Boundary condition equation: h*u=r

Condition type: Neumann [✓ Dirichlet]

Coefficient	Value	Description
g	0	
q	0	
h	1	
r	0	

Figure 5.2.1b: Boundary Condition Window

If we double click on an interface then the interface condition window pops up as presented in Figure 5.2.1c. In this window the user can insert the initial condition of the interface which can be a vector, a function or even another Matlab file which contains an expression that can be evaluated. The default interface relaxation method is GEO. There is also an option for a user define method. In this case the user has to create his / her own IR method.


Intearface Relaxation Condition

Interface Relaxation Equation: Geo

Condition type: User Define [✓ GEO]

Coefficient	Value	Description
Initial Condition	0	
l	0	
w	1	
p	0	

Figure 5.2.1c: Interface Relaxation Condition

5.2.2 PDE Menu

The PDE menu is exactly as it is in Matlab and has the following options as shown in Figure 5.2.2a: PDE Mode, Show Sub-domain Labels, PDE Specification and Export PDE Coefficients. The parameter d does not apply to the elliptic PDE. The coefficients a, c and f can be constants or functions. In order to specify the PDE equation and its coefficients for each subdomain we have to choose PDE Specification and enter the desired values in the window as shown in Figure 5.2.2b.

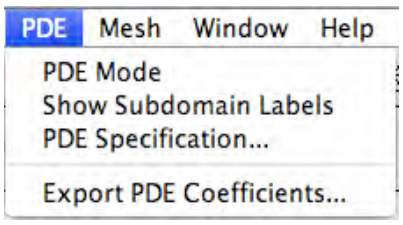
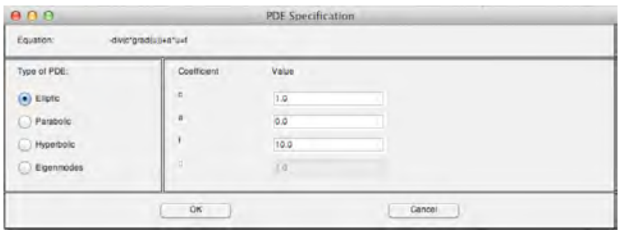


Figure 5.2.2a PDE Menu



Equation: -div(c*grad(u))+a*u=f

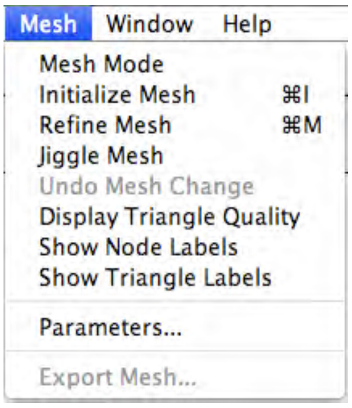
Type of PDE: [● Elliptic] ○ Parabolic ○ Hyperbolic ○ Eigenmodes

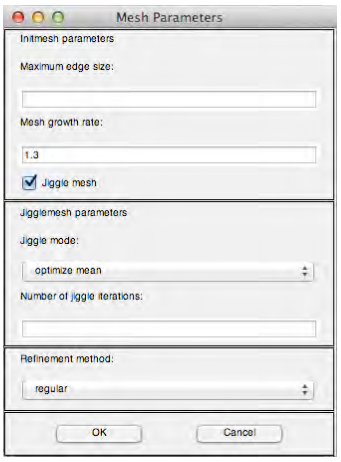
Coefficient	Value
c	1.0
a	0.0
f	10.0
d	1.0

Figure 5.2.2b: PDE Specification Window

5.2.3 Mesh Menu

The Mesh menu has the following options: Mesh Mode, Initialize Mesh, Re- fine Mesh, Jiggle Mesh, Undo Mesh Change, Display Triangle Quality, Show Node Labels, Show Triangles Labels, Parameters and Export Mesh as is presented in the Figure 5.2.3a. Parameters for controlling the jiggling of the mesh, the refinement method, and other mesh generation parameters can be found in a dialog box that is opened by selecting Parameters from the Mesh menu. At this time, we must initialize the maximum edge number, as this refers to the hmax. Hmax is necessary in order to solve the PDE problem. This menu is the same as it is in PDE Toolbox and the window that appears looks like the Figure 5.2.3b.





Initmesh parameters

Maximum edge size:

Mesh growth rate: 1.3

[✓ Jiggle mesh]

Jigglemesh parameters

Jiggle mode: optimize mean

Number of jiggle iterations:

Refinement method: regular

Figure 5.2.3a: Mesh Menu

Figure 5.2.3b: Mesh Parameters Window

Chapter 6: IRTool Implementation

6.1 General

In this chapter we are going to give further information about the implementation of IRToolbox. Since PDETool as has already been mentioned is open source we had to alter some of it's files in order to achieve the desirable result. We also had to create a few new files also to implement IRToolbox. The created files with all the code that has been written can be found at the end of this thesis in the appendix.

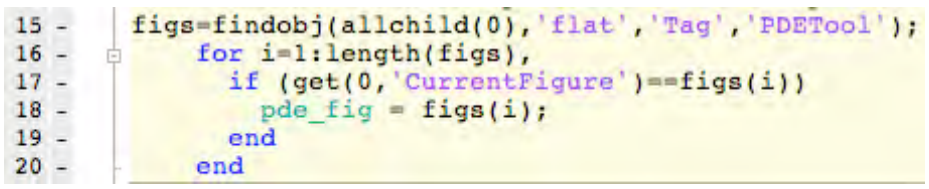
6.2 IRTool Initialization

One of the first changes that we had to make in Matlab's PDETool was its ability to open more than one window since only one at a time can be open. Since our primal problem has to be decomposed in many other, IRTool must support this functionality. As already presented in the previous chapter IRTool has a main / global window in which we draw the initial problem, and thereafter our toolbox decomposes it to each subproblems / subdomains.
The files that were created are

1. *irtool.m*
   It contains the function *irtool()* which simply calls the pdetool() function. This function was created to avoid writing *pdetool()* in order to initialize our toolbox. So in order ti initialize our tool we have to type ***irtool()***

The file that we altered was
1. *pdetool.m*

When pdetool is called with no input arguments it calls the pdetool('initialize') which draws the initial window. Before the drawing of the window we inserted the following lines of code.

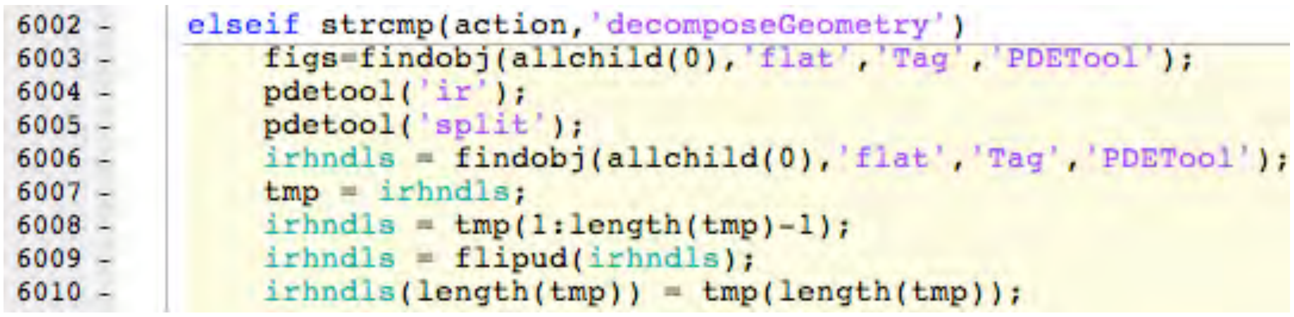


```
figs=findobj(allchild(0),'flat','Tag','PDETool');
  for i=1:length(figs),
    if (get(0,'CurrentFigure')==figs(i))
      pde_fig = figs(i);
    end
  end
```

It was also necessary to insert the global variable ***pde_fig*** which is used in almost every file which denotes at every given time the current IRTool window.

6.3 Geometry Decomposition

Once the user has drawn the geometry that describes his / her problem, he / she can then choose *Decompose Geometry* from the *Interface Relaxation* menu of the main / global window.
When the user chooses to decompose the initial / global geometry then the function *pdetool('decomposeGeometry')* is called. This function is an addition to pdetool.m file as another else - if statement for the kind of function that our toolbox wants to execute. The number of lines and the code of this block of code is presented bellow.

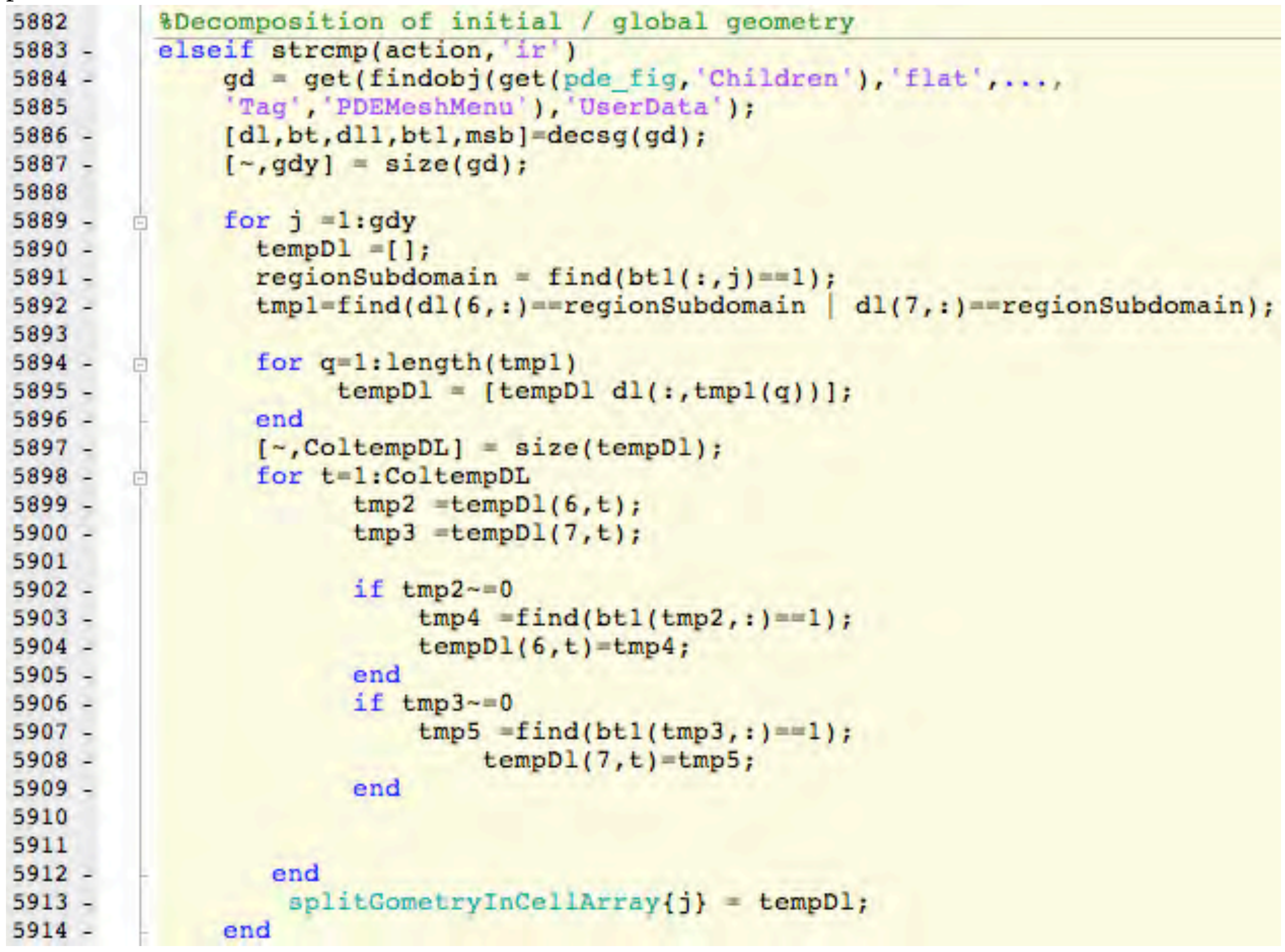


```
elseif strcmp(action,'decomposeGeometry')
    figs=findobj(allchild(0),'flat','Tag','PDETool');
    pdetool('ir');
    pdetool('split');
    irhndls = findobj(allchild(0),'flat','Tag','PDETool');
    tmp = irhndls;
    irhndls = tmp(1:length(tmp)-1);
    irhndls = flipud(irhndls);
    irhndls(length(tmp)) = tmp(length(tmp));
```

In this if -else branch two other functions all called which are also inside the *pdetool.m* file.

1. *pdetool('ir')*

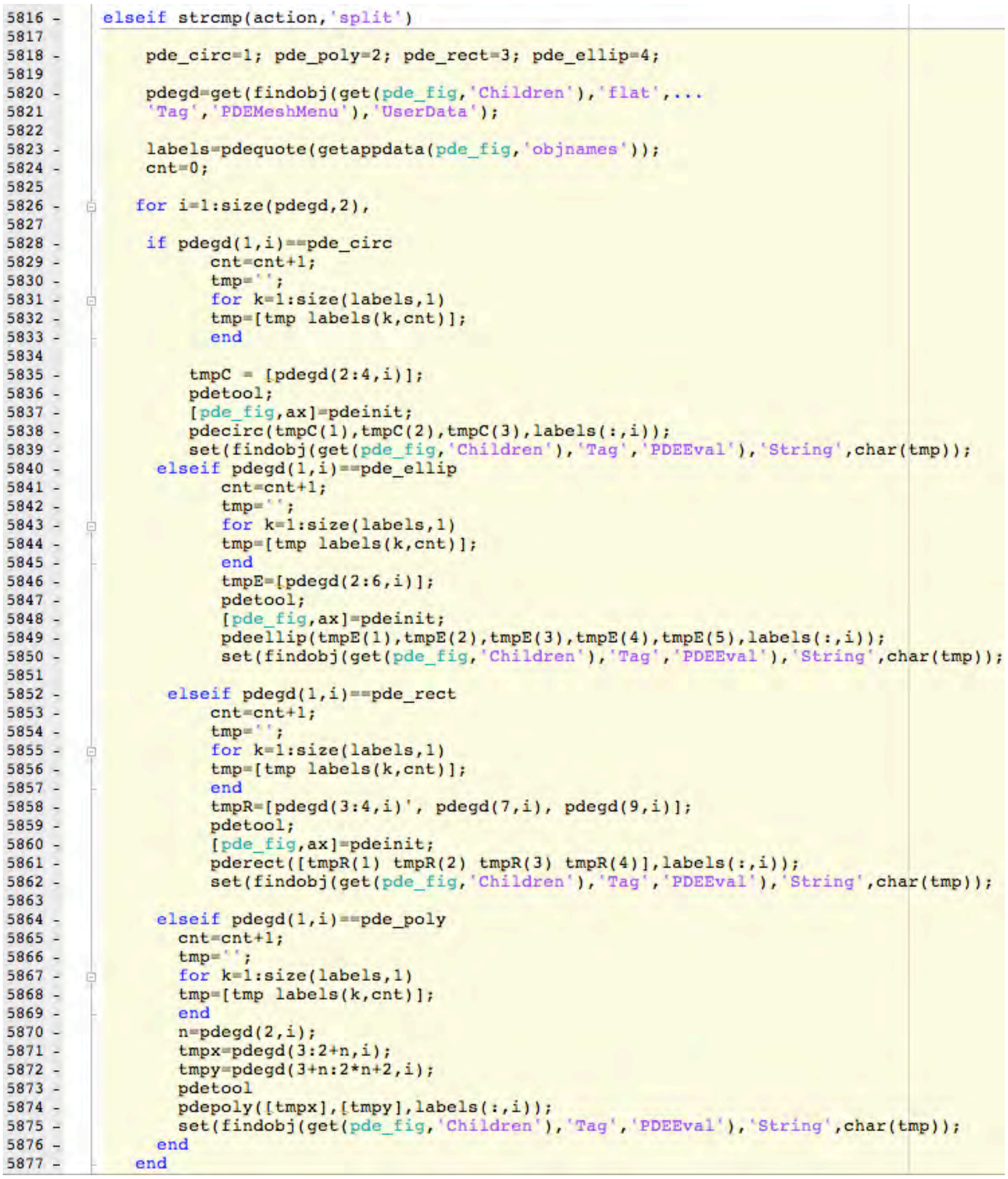
   This function finds the boundaries and the interfaces that every subdomain has. This is a necessary step before the decomposition of the geometry, since PDETool would ignore the interface segments. In this part of the development we had to introduce another global variable - a cell array - called *splitGometryInCellArray which contains the decomposed geometry of every subdomain.*The produced code is shown below



```
%Decomposition of initial / global geometry
elseif strcmp(action,'ir')
    gd = get(findobj(get(pde_fig,'Children'),'flat',...,
    'Tag','PDEMeshMenu'),'UserData');
    [dl,bt,dl1,bt1,msb]=decsg(gd);
    [~,gdy] = size(gd);

    for j =1:gdy
      tempDl =[];
      regionSubdomain = find(bt1(:,j)==1);
      tmp1=find(dl(6,:)==regionSubdomain | dl(7,:)==regionSubdomain);

      for q=1:length(tmp1)
          tempDl = [tempDl dl(:,tmp1(q))];
      end
      [~,ColtempDL] = size(tempDl);
      for t=1:ColtempDL
            tmp2 =tempDl(6,t);
            tmp3 =tempDl(7,t);

            if tmp2~=0
                tmp4 =find(bt1(tmp2,:)==1);
                tempDl(6,t)=tmp4;
            end
            if tmp3~=0
                tmp5 =find(bt1(tmp3,:)==1);
                    tempDl(7,t)=tmp5;
            end


       end
        splitGometryInCellArray{j} = tempDl;
    end
```

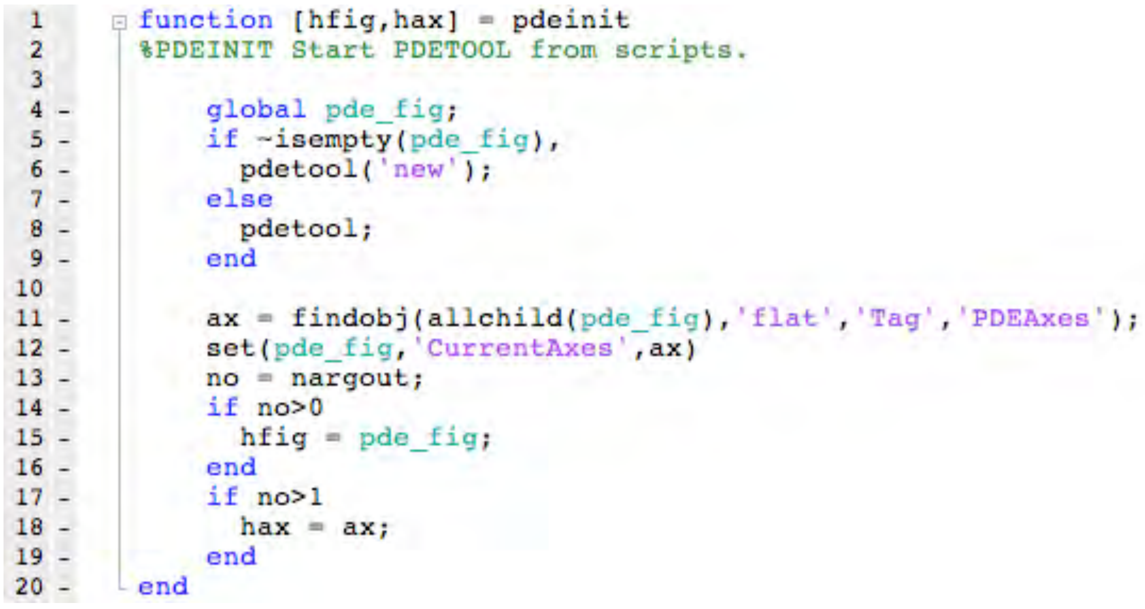
2. *pdetool('split')*

   In this code block the subdomain windows are created, since the geometry of every subdomain is now known by the previous function. At this part becomes more obvious the importance of the global variable *pde_fig* and the few lines of code that we inserted in the beginning of pdetool.m



```
elseif strcmp(action,'split')

    pde_circ=1; pde_poly=2; pde_rect=3; pde_ellip=4;

    pdegd=get(findobj(get(pde_fig,'Children'),'flat',...
    'Tag','PDEMeshMenu'),'UserData');

    labels=pdequote(getappdata(pde_fig,'objnames'));
    cnt=0;

   for i=1:size(pdegd,2),

    if pdegd(1,i)==pde_circ
          cnt=cnt+1;
          tmp='';
          for k=1:size(labels,1)
          tmp=[tmp labels(k,cnt)];
          end

        tmpC = [pdegd(2:4,i)];
        pdetool;
        [pde_fig,ax]=pdeinit;
        pdecirc(tmpC(1),tmpC(2),tmpC(3),labels(:,i));
        set(findobj(get(pde_fig,'Children'),'Tag','PDEEval'),'String',char(tmp));
     elseif pdegd(1,i)==pde_ellip
           cnt=cnt+1;
           tmp='';
           for k=1:size(labels,1)
           tmp=[tmp labels(k,cnt)];
           end
           tmpE=[pdegd(2:6,i)];
           pdetool;
           [pde_fig,ax]=pdeinit;
           pdeellip(tmpE(1),tmpE(2),tmpE(3),tmpE(4),tmpE(5),labels(:,i));
           set(findobj(get(pde_fig,'Children'),'Tag','PDEEval'),'String',char(tmp));

      elseif pdegd(1,i)==pde_rect
          cnt=cnt+1;
          tmp='';
          for k=1:size(labels,1)
          tmp=[tmp labels(k,cnt)];
          end
          tmpR=[pdegd(3:4,i)', pdegd(7,i), pdegd(9,i)];
          pdetool;
          [pde_fig,ax]=pdeinit;
          pderect([tmpR(1) tmpR(2) tmpR(3) tmpR(4)],labels(:,i));
          set(findobj(get(pde_fig,'Children'),'Tag','PDEEval'),'String',char(tmp));

     elseif pdegd(1,i)==pde_poly
       cnt=cnt+1;
       tmp='';
       for k=1:size(labels,1)
       tmp=[tmp labels(k,cnt)];
       end
       n=pdegd(2,i);
       tmpx=pdegd(3:2+n,i);
       tmpy=pdegd(3+n:2*n+2,i);
       pdetool
       pdepoly([tmpx],[tmpy],labels(:,i));
       set(findobj(get(pde_fig,'Children'),'Tag','PDEEval'),'String',char(tmp));
     end
   end
```

In the above code block pdeinit.m is called which is actually responsible for the kind of the initialization that happens. In PDEToolbox if there is not an opened window, this functions calls *pdetool.m* to create one. If there is an active window and pdeinit.m is called then it clears the current window. We had to change this function so that our tool will not clear the initial geometry but open another window. In the following code inside *pdeinit.m* we added lines 4 - 9.



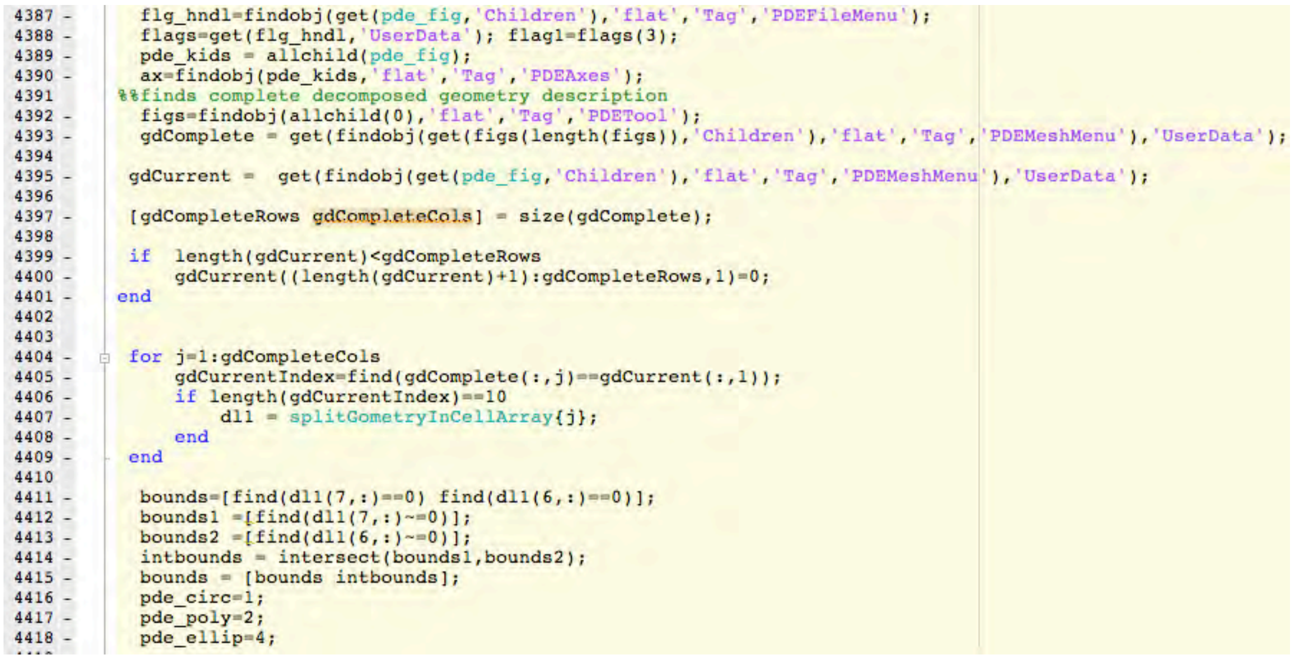
```
function [hfig,hax] = pdeinit
%PDEINIT Start PDETOOL from scripts.

    global pde_fig;
    if ~isempty(pde_fig),
      pdetool('new');
    else
      pdetool;
    end

    ax = findobj(allchild(pde_fig),'flat','Tag','PDEAxes');
    set(pde_fig,'CurrentAxes',ax)
    no = nargout;
    if no>0
      hfig = pde_fig;
    end
    if no>1
      hax = ax;
    end
end
```

6.4 Boundaries & Interfaces

Once the user has decomposed his / her geometry can enter boundary mode. We had to make a few modifications in this part too in order for the interfaces to appear in this mode. If we had not make the changes presented below our toolbox would handle interfaces as common boundaries. The changes we made in some existing files and the new files we created are the following:

- Changes
    1. pdetool('drawbounds')

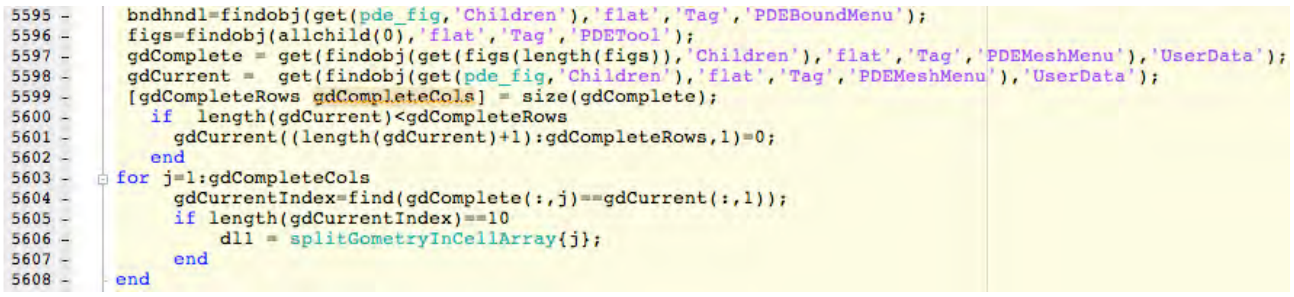
This function is called within *pdetool('boundmode')*. The decomposed geometry description matrix that PDETool creates for every subdomain contains information for all the line segments that consist a boundary / interface. The first row contains the type of segment, if it is a line,circle,ellipse segment. The next four lines contain information about the coordinates of each segment. The sixth and seventh lines contain information about the left and right hand region of each segment. This piece of information is vital for us. PDETool denotes the wide external region of a domain with zero. If both these lines are not both equal with zero means that is an interface. The additions / changes that we introduced are shown below.



```matlab
  flg_hndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEFileMenu');
  flags=get(flg_hndl,'UserData'); flag1=flags(3);
  pde_kids = allchild(pde_fig);
  ax=findobj(pde_kids,'flat','Tag','PDEAxes');
%%finds complete decomposed geometry description
  figs=findobj(allchild(0),'flat','Tag','PDETool');
  gdComplete = get(findobj(get(figs(length(figs)),'Children'),'flat','Tag','PDEMeshMenu'),'UserData');

 gdCurrent =  get(findobj(get(pde_fig,'Children'),'flat','Tag','PDEMeshMenu'),'UserData');

 [gdCompleteRows gdCompleteCols] = size(gdComplete);

 if  length(gdCurrent)<gdCompleteRows
     gdCurrent((length(gdCurrent)+1):gdCompleteRows,1)=0;
end


 for j=1:gdCompleteCols
     gdCurrentIndex=find(gdComplete(:,j)==gdCurrent(:,1));
     if length(gdCurrentIndex)==10
         dll = splitGometryInCellArray{j};
     end
 end

  bounds=[find(dll(7,:)==0) find(dll(6,:)==0)];
  bounds1 =[find(dll(7,:)~=0)];
  bounds2 =[find(dll(6,:)~=0)];
  intbounds = intersect(bounds1,bounds2);
  bounds = [bounds intbounds];
  pde_circ=1;
  pde_poly=2;
  pde_ellip=4;
```

2. *pdetool('initbounds')*

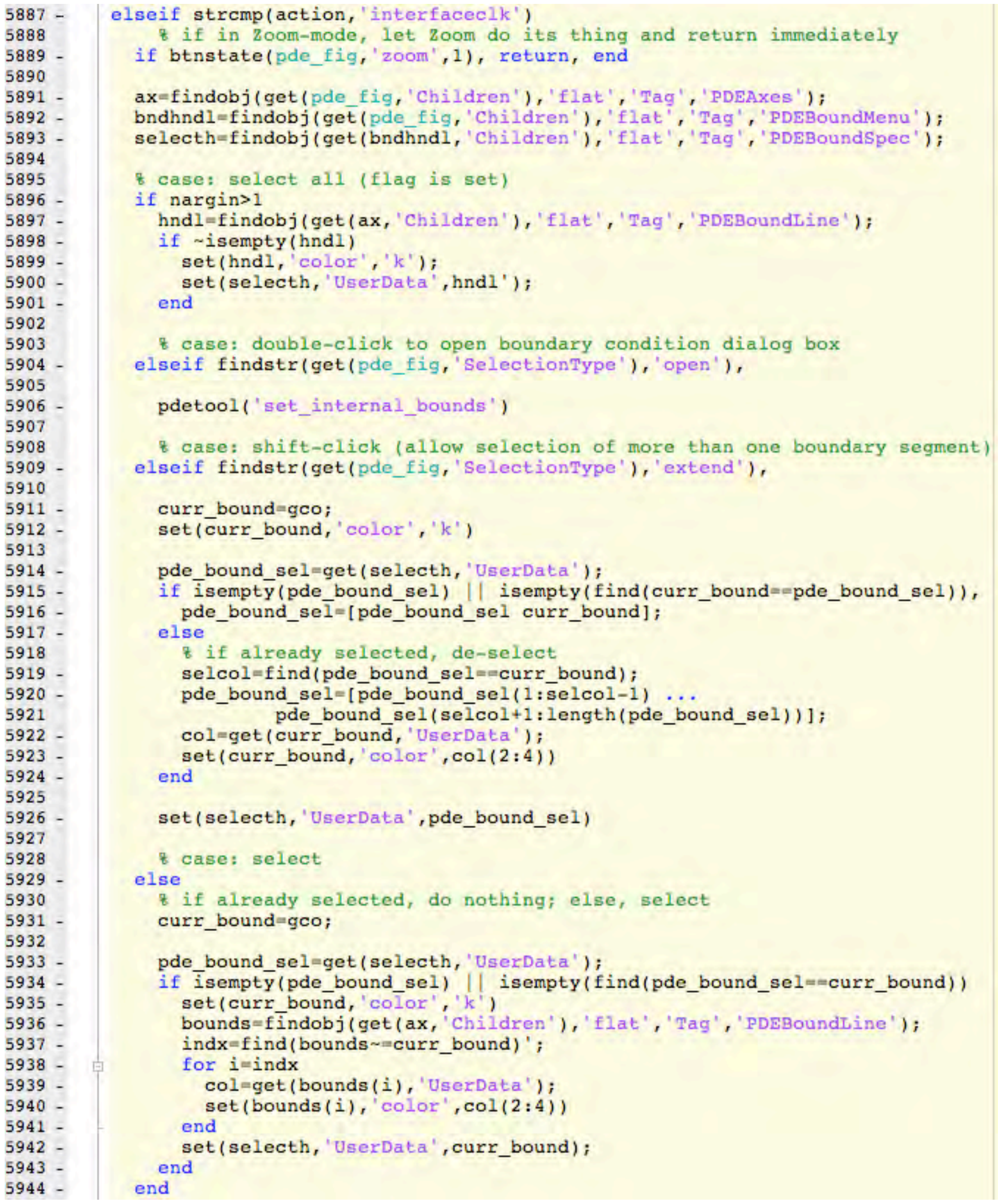
   It is called within *pdetool('drawbounds')* and packs and initializes the boundary and interface conditions.



```matlab
 bndhndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEBoundMenu');
 figs=findobj(allchild(0),'flat','Tag','PDETool');
 gdComplete = get(findobj(get(figs(length(figs)),'Children'),'flat','Tag','PDEMeshMenu'),'UserData');
 gdCurrent =  get(findobj(get(pde_fig,'Children'),'flat','Tag','PDEMeshMenu'),'UserData');
 [gdCompleteRows gdCompleteCols] = size(gdComplete);
   if  length(gdCurrent)<gdCompleteRows
     gdCurrent((length(gdCurrent)+1):gdCompleteRows,1)=0;
   end
for j=1:gdCompleteCols
     gdCurrentIndex=find(gdComplete(:,j)==gdCurrent(:,1));
     if length(gdCurrentIndex)==10
         dll = splitGometryInCellArray{j};
     end
end
```

3. *pdetool(interfaceclk)*

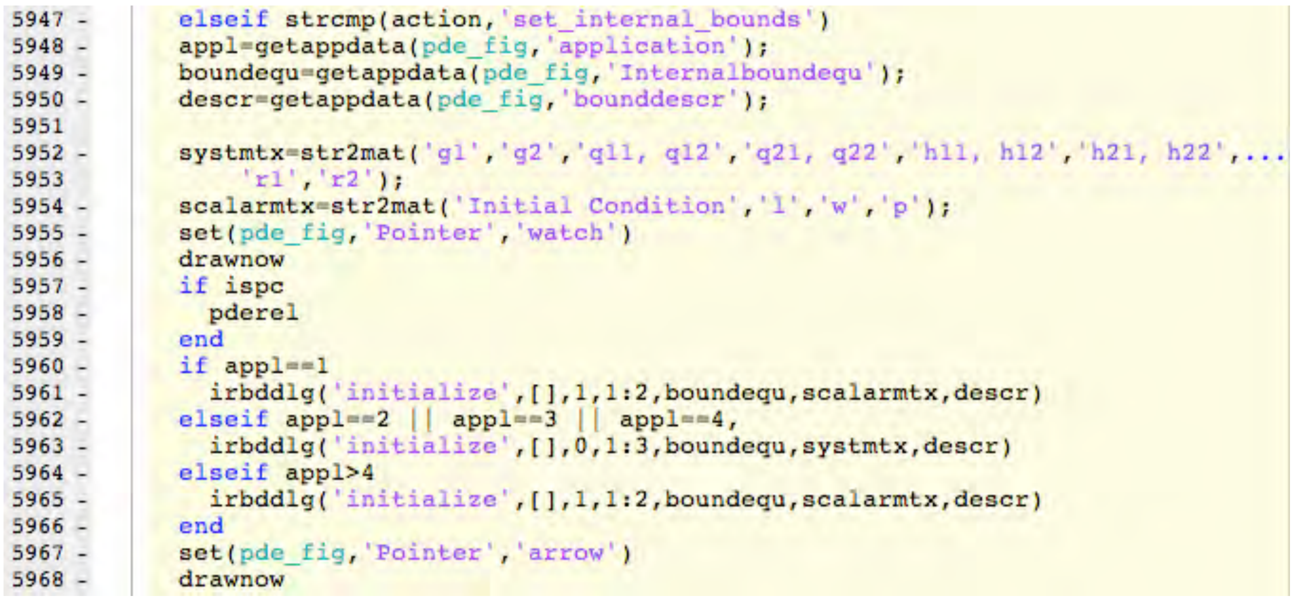
   It is another code block inside pdetool.m that makes us able to double click with the mouse on an interface in order to insert interface conditions.



```
elseif strcmp(action,'interfaceclk')
    % if in Zoom-mode, let Zoom do its thing and return immediately
  if btnstate(pde_fig,'zoom',1), return, end

  ax=findobj(get(pde_fig,'Children'),'flat','Tag','PDEAxes');
  bndhndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEBoundMenu');
  selecth=findobj(get(bndhndl,'Children'),'flat','Tag','PDEBoundSpec');

  % case: select all (flag is set)
  if nargin>1
    hndl=findobj(get(ax,'Children'),'flat','Tag','PDEBoundLine');
    if ~isempty(hndl)
      set(hndl,'color','k');
      set(selecth,'UserData',hndl');
    end

    % case: double-click to open boundary condition dialog box
  elseif findstr(get(pde_fig,'SelectionType'),'open'),

    pdetool('set_internal_bounds')

    % case: shift-click (allow selection of more than one boundary segment)
  elseif findstr(get(pde_fig,'SelectionType'),'extend'),

    curr_bound=gco;
    set(curr_bound,'color','k')

    pde_bound_sel=get(selecth,'UserData');
    if isempty(pde_bound_sel) || isempty(find(curr_bound==pde_bound_sel)),
      pde_bound_sel=[pde_bound_sel curr_bound];
    else
      % if already selected, de-select
      selcol=find(pde_bound_sel==curr_bound);
      pde_bound_sel=[pde_bound_sel(1:selcol-1) ...
              pde_bound_sel(selcol+1:length(pde_bound_sel))];
      col=get(curr_bound,'UserData');
      set(curr_bound,'color',col(2:4))
    end

    set(selecth,'UserData',pde_bound_sel)

    % case: select
  else
    % if already selected, do nothing; else, select
    curr_bound=gco;

    pde_bound_sel=get(selecth,'UserData');
    if isempty(pde_bound_sel) || isempty(find(pde_bound_sel==curr_bound))
      set(curr_bound,'color','k')
      bounds=findobj(get(ax,'Children'),'flat','Tag','PDEBoundLine');
      indx=find(bounds~=curr_bound)';
      for i=indx
        col=get(bounds(i),'UserData');
        set(bounds(i),'color',col(2:4))
      end
      set(selecth,'UserData',curr_bound);
    end
  end
```

4. pdetool('set_internal_bounds')

Its called from *pdetool(interfaceclk)* and is responsible for preparing the interface to acquire the input that the user enters - pack and unpack parameters-.



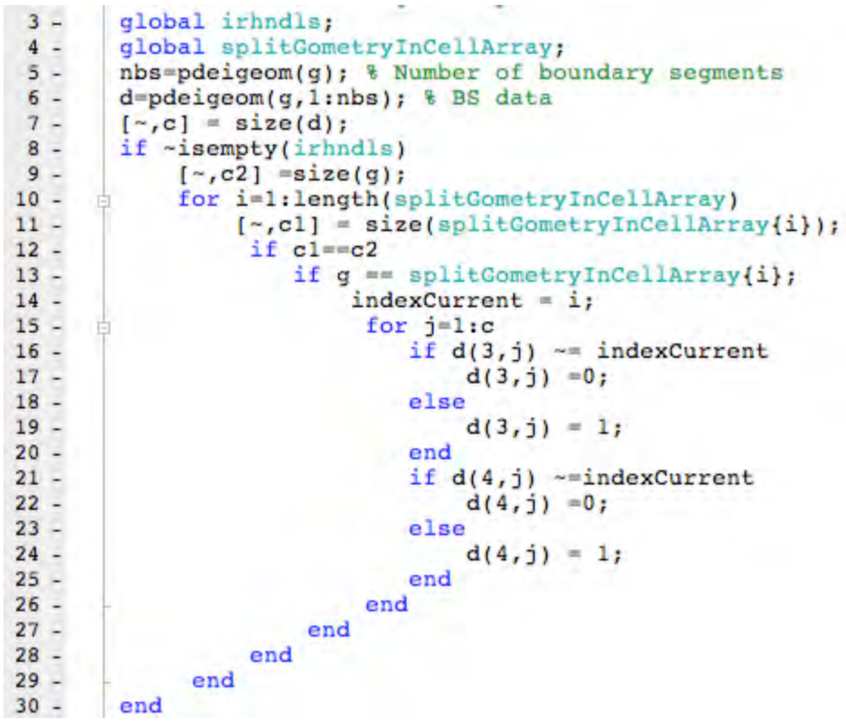
```
elseif strcmp(action,'set_internal_bounds')
appl=getappdata(pde_fig,'application');
boundequ=getappdata(pde_fig,'Internalboundequ');
descr=getappdata(pde_fig,'bounddescr');

systmtx=str2mat('g1','g2','q11, q12','q21, q22','h11, h12','h21, h22',...
    'r1','r2');
scalarmtx=str2mat('Initial Condition','1','w','p');
set(pde_fig,'Pointer','watch')
drawnow
if ispc
  pderel
end
if appl==1
  irbddlg('initialize',[],1,1:2,boundequ,scalarmtx,descr)
elseif appl==2 || appl==3 || appl==4,
  irbddlg('initialize',[],0,1:3,boundequ,systmtx,descr)
elseif appl>4
  irbddlg('initialize',[],1,1:2,boundequ,scalarmtx,descr)
end
set(pde_fig,'Pointer','arrow')
drawnow
```

- New file created
    1. *irbddlg.m*

This function designs and handles the interface boundary condition window that appears when the user double clicks with his / her mouse on an interface line. The code implementation of this function can be found in the appendix.

6.5 Mesh Initialization

One other change that we had to make in order our toolbox to work properly is the mesh production strategy. Once the initial geometry has been decomposed to component subdomains the mesh generation of each geometry changes. This is forced by the fact that since each subdomain is single on each window it does not have at the left / right hand region in the decomposed geometry matrix one or more segments with other neighboring subdomain / subdomains. So if we proceed to create the mesh for each subdomain, a wrong one will be produced. In order to achieve the correct mesh initialization we had to alter the function *pdemgeom.m* which is called from the function *initmesh.m* that starts the mesh creation. Specifically after the decomposition of each subdomain we search which line segments have a left / right hand region that is not the same with the internal region of the current subdomain. Once this region is found is set to zero. This change of values is not saved in the decomposed geometry matrix-is only used for the triangulation. The code block that has been added to *pdemgeon.m* is

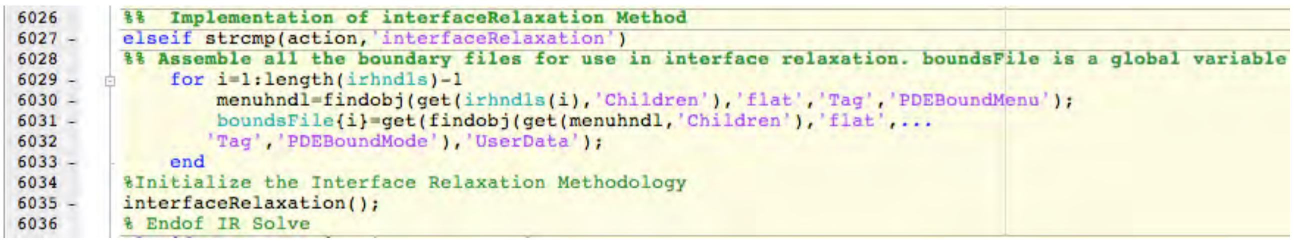


```
global irhndls;
global splitGometryInCellArray;
nbs=pdeigeom(g); % Number of boundary segments
d=pdeigeom(g,1:nbs); % BS data
[~,c] = size(d);
if ~isempty(irhndls)
    [~,c2] =size(g);
    for i=1:length(splitGometryInCellArray)
        [~,c1] = size(splitGometryInCellArray{i});
         if c1==c2
            if g == splitGometryInCellArray{i};
                indexCurrent = i;
                 for j=1:c
                    if d(3,j) ~= indexCurrent
                        d(3,j) =0;
                    else
                        d(3,j) = 1;
                    end
                    if d(4,j) ~=indexCurrent
                        d(4,j) =0;
                    else
                        d(4,j) = 1;
                    end
                end
            end
        end
    end
end
```

6.6 Interface Relaxation

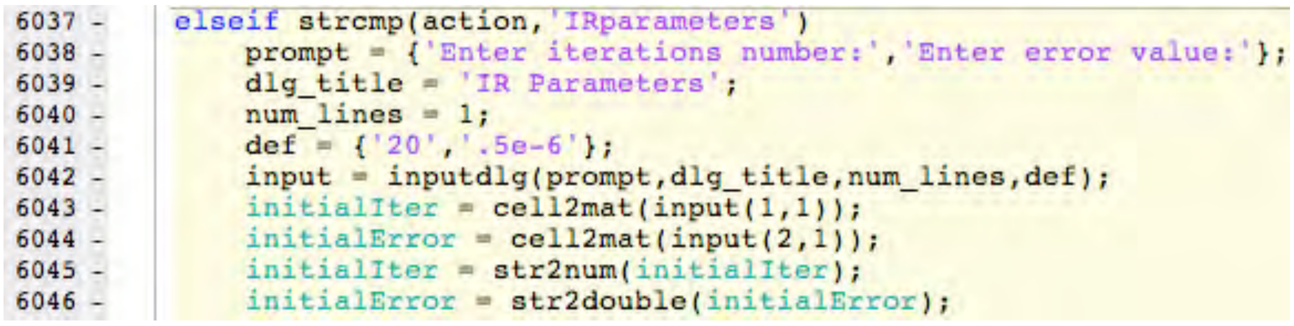
The last stage of the development of our toolbox was the implementation of the interface relaxation method. At this stage appeared any other file modifications that we had to implement and also the creation of many new functions. Due to the fact that all the files that were created were extensive in code lines here we will present only the line codes of *interfaceRelaxation.m* function as long as the functionality of all the functions that are called by it. The implementation of those functions can be found in the Appendix.

Firstly we had to add a few lines of code in the *pdetool.m function* in order to make the menu Solve MultiPDE responsive and functional.



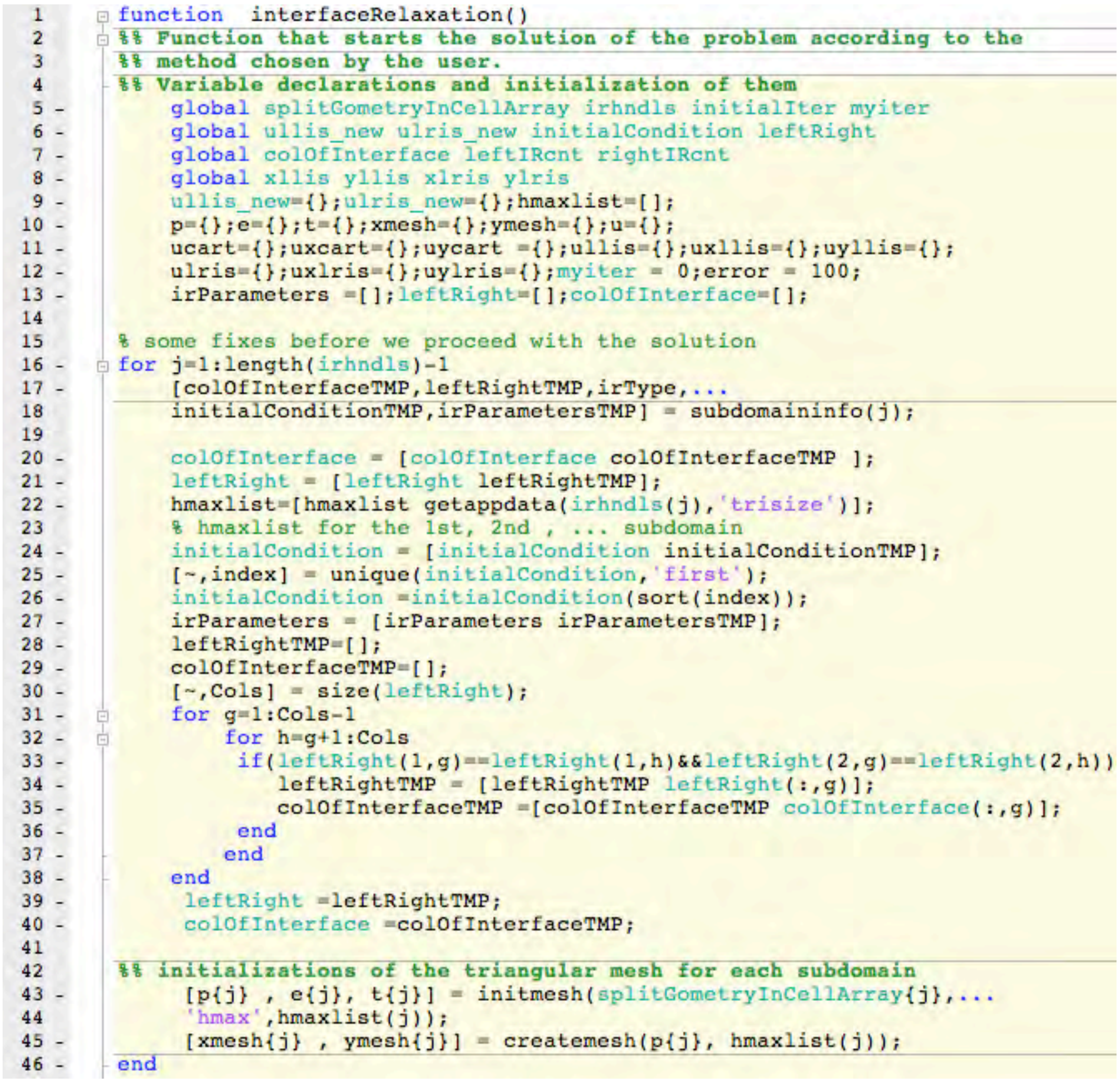
```
%%  Implementation of interfaceRelaxation Method
elseif strcmp(action,'interfaceRelaxation')
%% Assemble all the boundary files for use in interface relaxation. boundsFile is a global variable
    for i=1:length(irhndls)-1
        menuhndl=findobj(get(irhndls(i),'Children'),'flat','Tag','PDEBoundMenu');
        boundsFile{i}=get(findobj(get(menuhndl,'Children'),'flat',...
       'Tag','PDEBoundMode'),'UserData');
    end
%Initialize the Interface Relaxation Methodology
interfaceRelaxation();
% Endof IR Solve
```

Additionally we had to implement also the functionality of the pop up window for the user input of iteration number and error for the interface relaxation methodology.

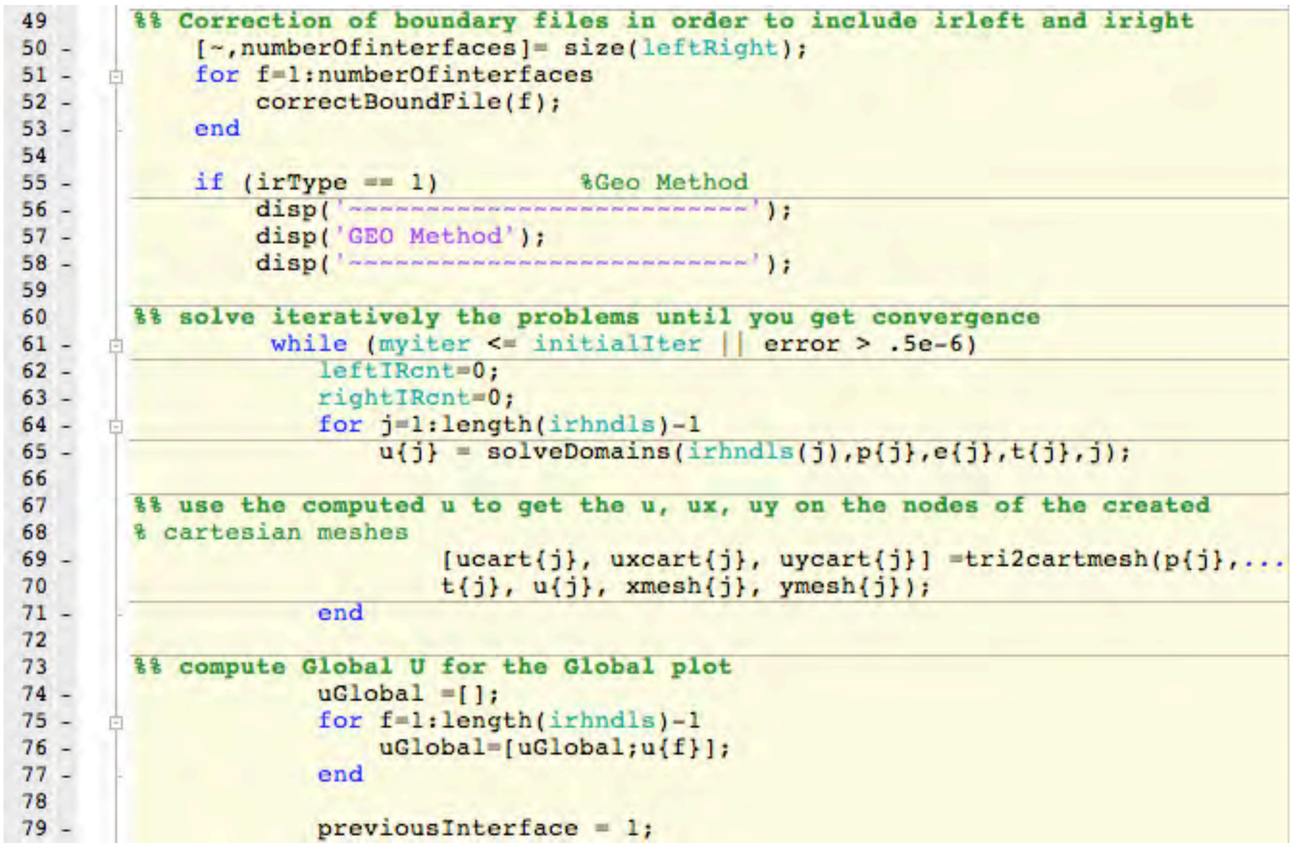


```
    elseif strcmp(action,'IRparameters')
        prompt = {'Enter iterations number:','Enter error value:'};
        dlg_title = 'IR Parameters';
        num_lines = 1;
        def = {'20','.5e-6'};
        input = inputdlg(prompt,dlg_title,num_lines,def);
        initialIter = cell2mat(input(1,1));
        initialError = cell2mat(input(2,1));
        initialIter = str2num(initialIter);
        initialError = str2double(initialError);
```

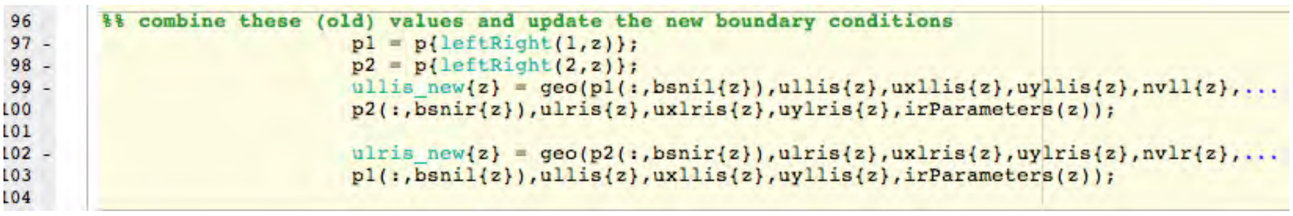
Once the user has finished entering all the necessary variables of his / her problem can start the solution of the problem by pressing the Solve MultiPDE choice from the Interface Relaxation Menu on the main / global window. This action calls the *interfaceRelaxation.m* function which is presented below



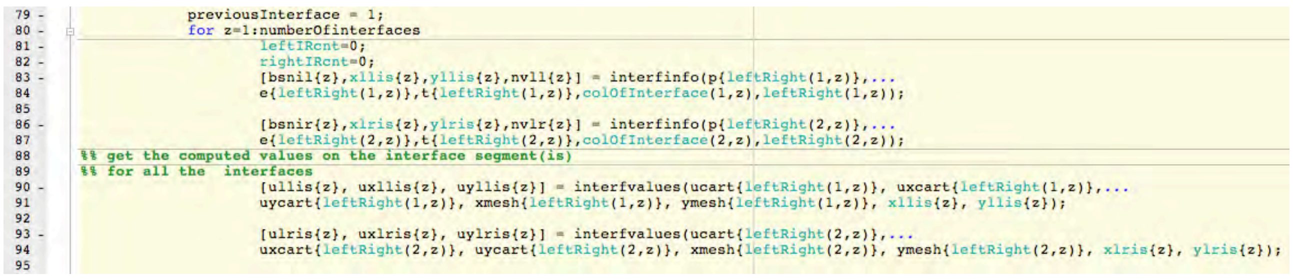
```
function  interfaceRelaxation()
%% Function that starts the solution of the problem according to the
%% method chosen by the user.
%% Variable declarations and initialization of them
    global splitGometryInCellArray irhndls initialIter myiter
    global ullis_new ulris_new initialCondition leftRight
    global colOfInterface leftIRcnt rightIRcnt
    global xllis yllis xlris ylris
    ullis_new={};ulris_new={};hmaxlist=[];
    p={};e={};t={};xmesh={};ymesh={};u={};
    ucart={};uxcart={};uycart ={};ullis={};uxllis={};uyllis={};
    ulris={};uxlris={};uylris={};myiter = 0;error = 100;
    irParameters =[];leftRight=[];colOfInterface=[];

% some fixes before we proceed with the solution
for j=1:length(irhndls)-1
    [colOfInterfaceTMP,leftRightTMP,irType,...
    initialConditionTMP,irParametersTMP] = subdomaininfo(j);

    colOfInterface = [colOfInterface colOfInterfaceTMP ];
    leftRight = [leftRight leftRightTMP];
    hmaxlist=[hmaxlist getappdata(irhndls(j),'trisize')];
    % hmaxlist for the 1st, 2nd , ... subdomain
    initialCondition = [initialCondition initialConditionTMP];
    [~,index] = unique(initialCondition,'first');
    initialCondition =initialCondition(sort(index));
    irParameters = [irParameters irParametersTMP];
    leftRightTMP=[];
    colOfInterfaceTMP=[];
    [~,Cols] = size(leftRight);
    for g=1:Cols-1
        for h=g+1:Cols
         if(leftRight(1,g)==leftRight(1,h)&&leftRight(2,g)==leftRight(2,h))
            leftRightTMP = [leftRightTMP leftRight(:,g)];
            colOfInterfaceTMP =[colOfInterfaceTMP colOfInterface(:,g)];
         end
        end
    end
     leftRight =leftRightTMP;
     colOfInterface =colOfInterfaceTMP;

%% initializations of the triangular mesh for each subdomain
     [p{j} , e{j}, t{j}] = initmesh(splitGometryInCellArray{j},...
     'hmax',hmaxlist(j));
     [xmesh{j} , ymesh{j}] = createmesh(p{j}, hmaxlist(j));
end
```



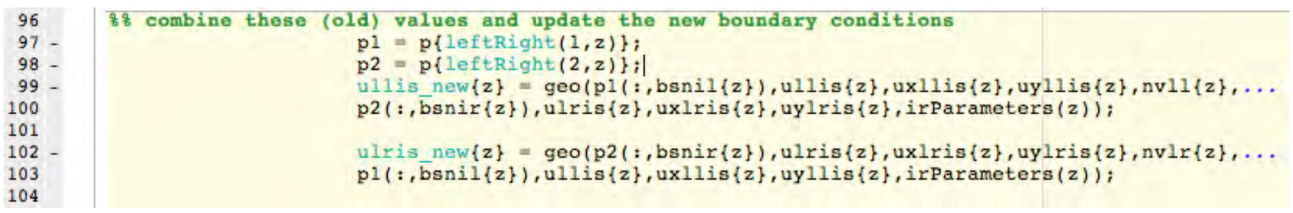
```
%% Correction of boundary files in order to include irleft and iright
    [~,numberOfinterfaces]= size(leftRight);
    for f=1:numberOfinterfaces
        correctBoundFile(f);
    end

    if (irType == 1)          %Geo Method
        disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~');
        disp('GEO Method');
        disp('~~~~~~~~~~~~~~~~~~~~~~~~~~~~');

%% solve iteratively the problems until you get convergence
         while (myiter <= initialIter || error > .5e-6)
            leftIRcnt=0;
            rightIRcnt=0;
            for j=1:length(irhndls)-1
                u{j} = solveDomains(irhndls(j),p{j},e{j},t{j},j);

%% use the computed u to get the u, ux, uy on the nodes of the created
% cartesian meshes
                [ucart{j}, uxcart{j}, uycart{j}] =tri2cartmesh(p{j},...
                t{j}, u{j}, xmesh{j}, ymesh{j});
            end

%% compute Global U for the Global plot
            uGlobal =[];
            for f=1:length(irhndls)-1
                uGlobal=[uGlobal;u{f}];
            end

            previousInterface = 1;
```



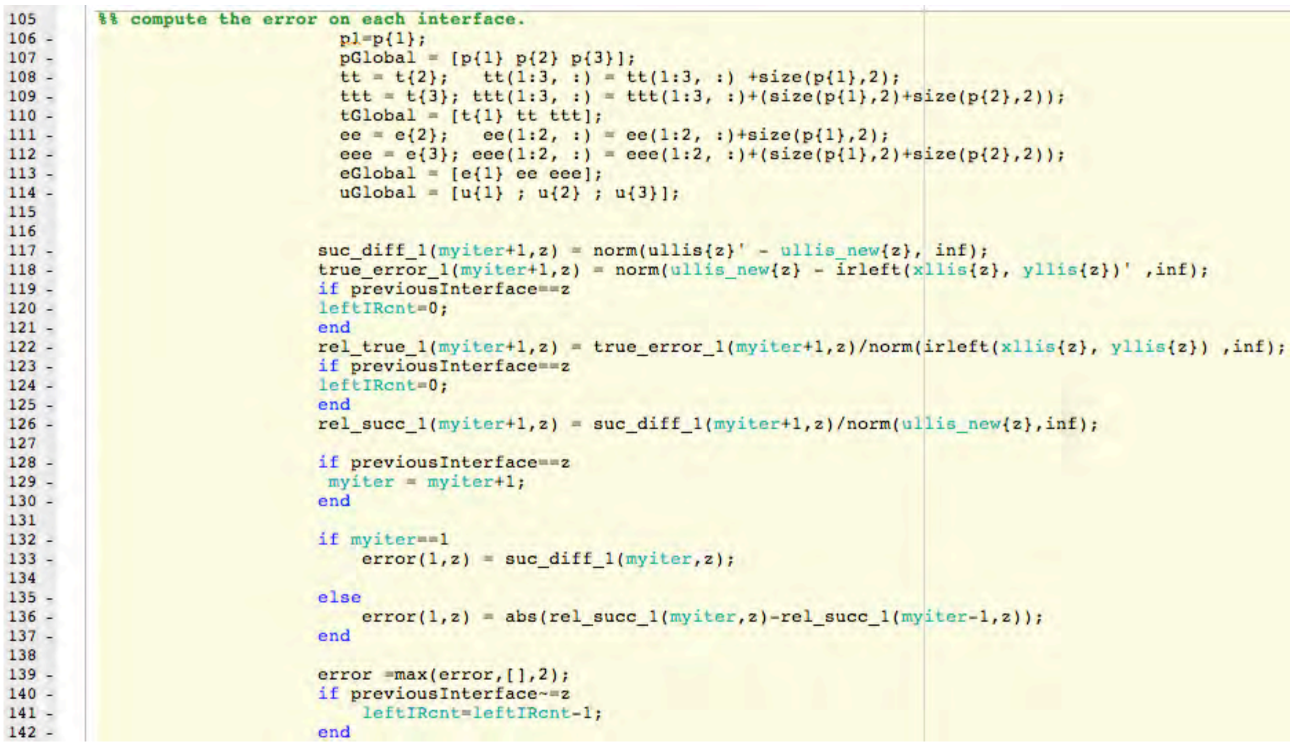
```
%% combine these (old) values and update the new boundary conditions
                p1 = p{leftRight(1,z)};
                p2 = p{leftRight(2,z)};
                ullis_new{z} = geo(p1(:,bsnil{z}),ullis{z},uxllis{z},uyllis{z},nvll{z},...
                p2(:,bsnir{z}),ulris{z},uxlris{z},uylris{z},irParameters(z));

                ulris_new{z} = geo(p2(:,bsnir{z}),ulris{z},uxlris{z},uylris{z},nvlr{z},...
                p1(:,bsnil{z}),ullis{z},uxllis{z},uyllis{z},irParameters(z));
```



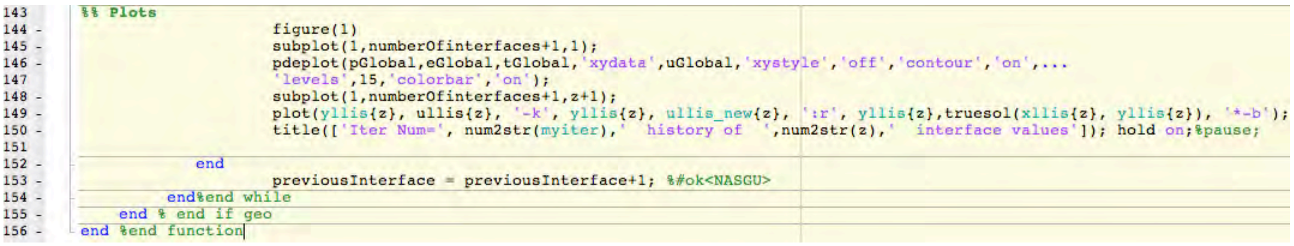
```
            previousInterface = 1;
            for z=1:numberOfinterfaces
                leftIRcnt=0;
                rightIRcnt=0;
                [bsnil{z},xllis{z},yllis{z},nvll{z}] = interfinfo(p{leftRight(1,z)},...
                e{leftRight(1,z)},t{leftRight(1,z)},colOfInterface(1,z),leftRight(1,z));

                [bsnir{z},xlris{z},ylris{z},nvlr{z}] = interfinfo(p{leftRight(2,z)},...
                e{leftRight(2,z)},t{leftRight(2,z)},colOfInterface(2,z),leftRight(2,z));
%% get the computed values on the interface segment(is)
%% for all the  interfaces
                [ullis{z}, uxllis{z}, uyllis{z}] = interfvalues(ucart{leftRight(1,z)}, uxcart{leftRight(1,z)},...
                uycart{leftRight(1,z)}, xmesh{leftRight(1,z)}, ymesh{leftRight(1,z)}, xllis{z}, yllis{z});

                [ulris{z}, uxlris{z}, uylris{z}] = interfvalues(ucart{leftRight(2,z)},...
                uxcart{leftRight(2,z)}, uycart{leftRight(2,z)}, xmesh{leftRight(2,z)}, ymesh{leftRight(2,z)}, xlris{z}, ylris{z});
```



```
%% combine these (old) values and update the new boundary conditions
                p1 = p{leftRight(1,z)};
                p2 = p{leftRight(2,z)};
                ullis_new{z} = geo(p1(:,bsnil{z}),ullis{z},uxllis{z},uyllis{z},nvll{z},...
                p2(:,bsnir{z}),ulris{z},uxlris{z},uylris{z},irParameters(z));

                ulris_new{z} = geo(p2(:,bsnir{z}),ulris{z},uxlris{z},uylris{z},nvlr{z},...
                p1(:,bsnil{z}),ullis{z},uxllis{z},uyllis{z},irParameters(z));
```



```
%% compute the error on each interface.
                        p1=p{1};
                        pGlobal = [p{1} p{2} p{3}];
                        tt = t{2};   tt(1:3, :) = tt(1:3, :) +size(p{1},2);
                        ttt = t{3}; ttt(1:3, :) = ttt(1:3, :)+(size(p{1},2)+size(p{2},2));
                        tGlobal = [t{1} tt ttt];
                        ee = e{2};   ee(1:2, :) = ee(1:2, :)+size(p{1},2);
                        eee = e{3}; eee(1:2, :) = eee(1:2, :)+(size(p{1},2)+size(p{2},2));
                        eGlobal = [e{1} ee eee];
                        uGlobal = [u{1} ; u{2} ; u{3}];


                      suc_diff_l(myiter+1,z) = norm(ullis{z}' - ullis_new{z}, inf);
                      true_error_l(myiter+1,z) = norm(ullis_new{z} - irleft(xllis{z}, yllis{z})' ,inf);
                      if previousInterface==z
                      leftIRcnt=0;
                      end
                      rel_true_l(myiter+1,z) = true_error_l(myiter+1,z)/norm(irleft(xllis{z}, yllis{z}) ,inf);
                      if previousInterface==z
                      leftIRcnt=0;
                      end
                      rel_succ_l(myiter+1,z) = suc_diff_l(myiter+1,z)/norm(ullis_new{z},inf);

                      if previousInterface==z
                       myiter = myiter+1;
                      end

                      if myiter==1
                          error(1,z) = suc_diff_l(myiter,z);

                      else
                          error(1,z) = abs(rel_succ_l(myiter,z)-rel_succ_l(myiter-1,z));
                      end

                      error =max(error,[],2);
                      if previousInterface~=z
                          leftIRcnt=leftIRcnt-1;
                      end
```



```
%% Plots
                        figure(1)
                        subplot(1,numberOfinterfaces+1,1);
                        pdeplot(pGlobal,eGlobal,tGlobal,'xydata',uGlobal,'xystyle','off','contour','on',...
                        'levels',15,'colorbar','on');
                        subplot(1,numberOfinterfaces+1,z+1);
                        plot(yllis{z}, ullis{z}, '-k', yllis{z}, ullis_new{z}, ':r', yllis{z},truesol(xllis{z}, yllis{z}), '*-b');
                        title(['Iter Num=', num2str(myiter),'  history of  ',num2str(z),'  interface values']); hold on;%pause;

                end
                        previousInterface = previousInterface+1; %#ok<NASGU>
             end%end while
        end % end if geo
end %end function
```

Inside the interfaceRelaxation function the following functions are being called:

1. *subdomaininfo.m*
   Gathers all the specifications and variables that we inserted for each subdomain, such as IR parameters, IR initial condition, the line segments in which an interface exists e.t.c.,

2. *initmesh.m*
   Builds an initial triangular PDE mesh

3. *createmesh.m*
   Creates a cartesian mesh

4. *correctBoundFile.m*
   After the extraction of the interfaces initial conditions it replaces them with irleft or iright according to whether the current interface is in the left or right hand region of the domain under inspection

5. *solveDomains.m*
   Solves the domain in order to get the new value for every iteration

6. *tri2cartmesh.m*
   Computes uc and its derivatives (Uxc and Uyx) on the cartesian

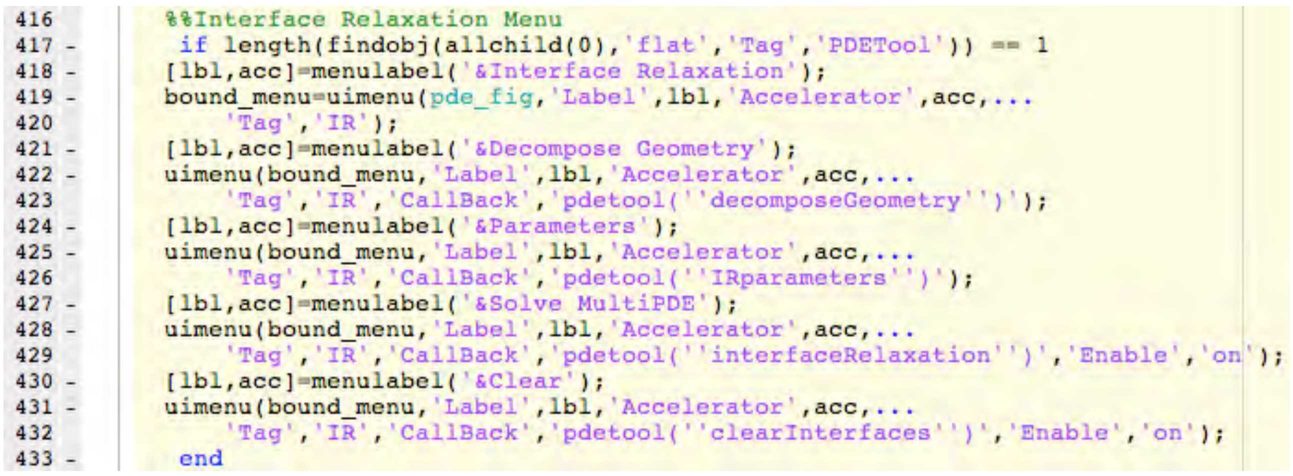
7. *interfinfo.m*
   Given a mesh of a domain, find out information for a specific boundary segment. This information is valuable for the IR methodology.

8. *interfvalues.m*
   Computes the function and its derivatives on the interface points

9. *geo.m*
   computes the new values of u on the interface nodes of
   each domain relaxing the old values from both neighboring domains
   using the outward normal vector .

6.7 Graphical User Interface (GUI)

Last but not least we had to make some changes in the initial and the subdomain window layout. These changes concern if a menu button is available at any stage of the problem modeling procedure. The changes were made in the *pdetool.m* file and are inflicted in lines 134 - 433. We present here only the code block for the *Interface Relaxation* menu which was not implemented in PDEToolbox.



```
416       %%Interface Relaxation Menu
417 -      if length(findobj(allchild(0),'flat','Tag','PDETool')) == 1
418 -     [lbl,acc]=menulabel('&Interface Relaxation');
419 -     bound_menu=uimenu(pde_fig,'Label',lbl,'Accelerator',acc,...
420           'Tag','IR');
421 -     [lbl,acc]=menulabel('&Decompose Geometry');
422 -     uimenu(bound_menu,'Label',lbl,'Accelerator',acc,...
423           'Tag','IR','CallBack','pdetool(''decomposeGeometry'')');
424 -     [lbl,acc]=menulabel('&Parameters');
425 -     uimenu(bound_menu,'Label',lbl,'Accelerator',acc,...
426           'Tag','IR','CallBack','pdetool(''IRparameters'')');
427 -     [lbl,acc]=menulabel('&Solve MultiPDE');
428 -     uimenu(bound_menu,'Label',lbl,'Accelerator',acc,...
429           'Tag','IR','CallBack','pdetool(''interfaceRelaxation'')','Enable','on');
430 -     [lbl,acc]=menulabel('&Clear');
431 -     uimenu(bound_menu,'Label',lbl,'Accelerator',acc,...
432           'Tag','IR','CallBack','pdetool(''clearInterfaces'')','Enable','on');
433 -      end
```

Chapter 7: User Guide / Numerical Experiments

7.1 General

The purpose of this chapter is to demonstrate and verify the functionality of *IRTool* by experimentally examining its validity. For this dissertation we have implemented the GEO method for elliptic problems using *IRTool* we restricted our experiments to partitions in two or three subdomains.

7.2 Two - dimensional elliptic partial differential problem

7.2.1 Uniform Problem - Solution Step By Step

For this let us consider the following elliptic problem:

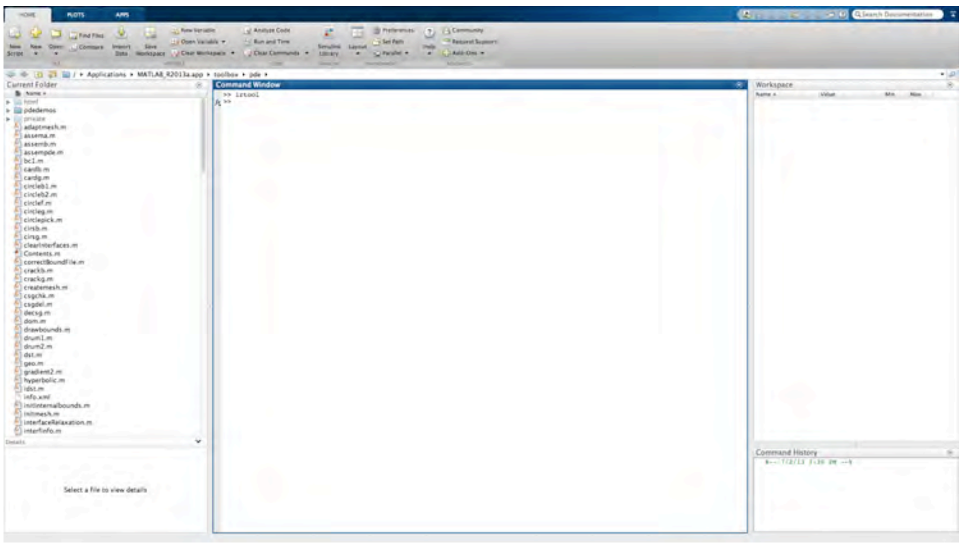
$$Lu(x,y) \equiv -\nabla^2 u(x,y)+\gamma^2 u(x,y)=f(x,y),\ (x,y) \in \Omega$$
$$u(x, y) = u^b(x, y),\ (x, y) \in \partial\Omega,$$

The true solution *u(x,y)* is

$$u(x,y) = e^{y(x+4)}x(x-1)(x-0.9)y(y-0.5).$$

This PDE problem consists of a geometry that is decomposed into three domains with interfaces on x_1 = 1/3 and x_2 = 2/3, Hmax =0.05 and omega is 0.03 and 0.04 for the first and second interface respectively.

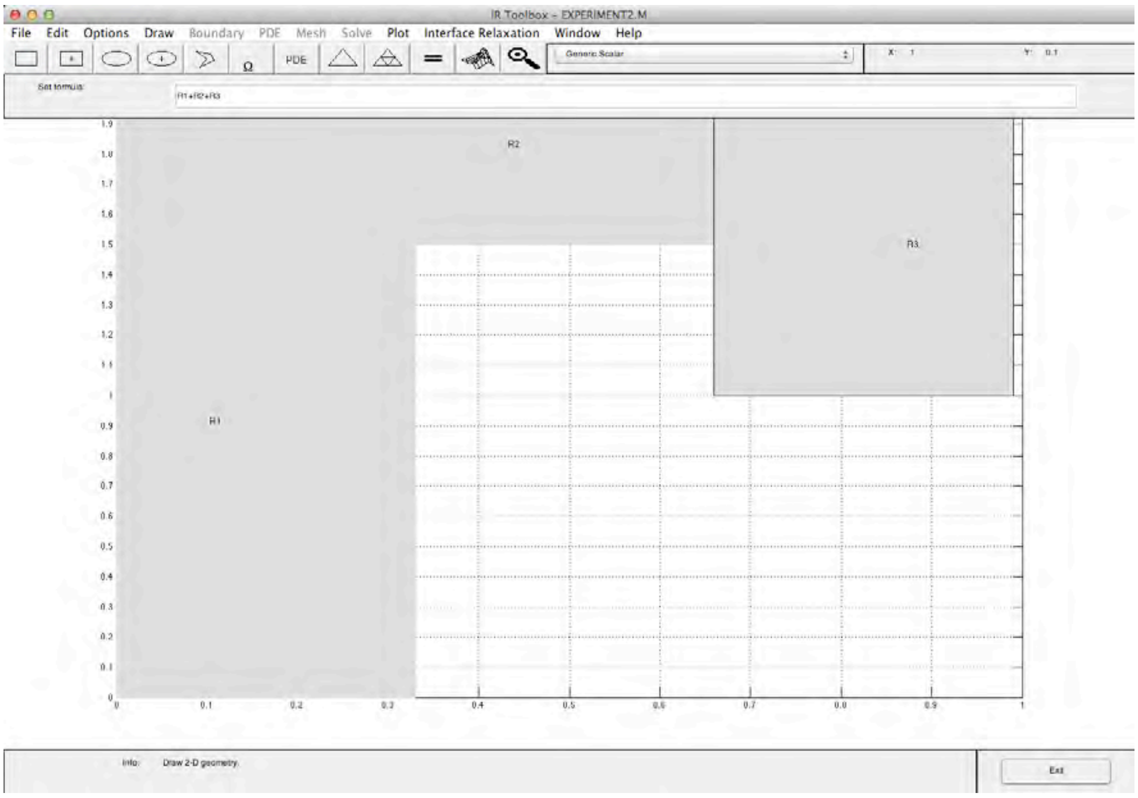
In order to initialize *IRTool* we type ***irtool*** in Matlab’s command window.



```
>> irtool
>>
```

Figure 7.1: Matlab’s Command Window

Automatically *IRTool* enters *Draw Mode*. In this stage the user has to draw the physical problem. One can either use the *Draw Menu* or the buttons below the menu in order to determine what type of geometry his / her problem consists of.



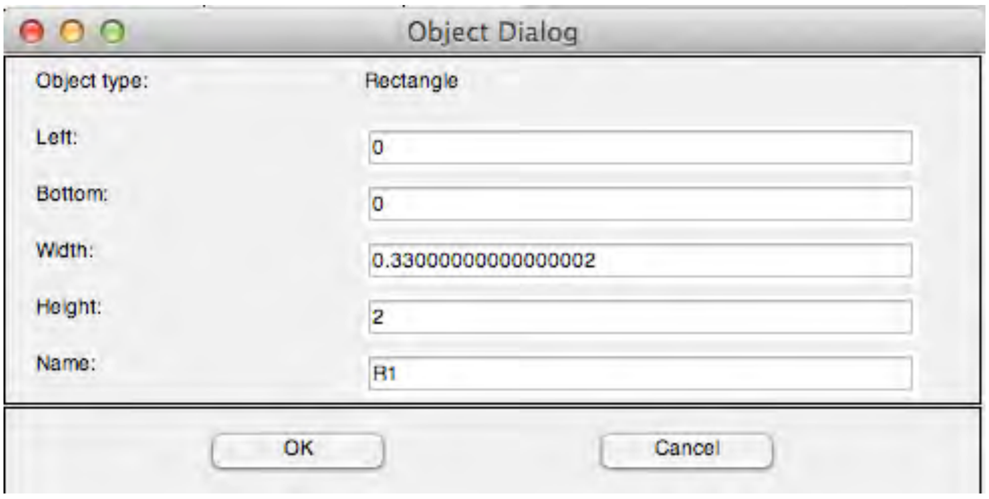
Set formula: R1+R2+R3

X: 1 Y: 0.1

Info: Draw 2-D geometry.

Figure 7.2: Geometry of the uniform problem

In order to achieve maximum accuracy for our domains we can double click on any subdomain and manually insert the coordinates, height and width of each one. The window that appears is shown below



Object type:	Rectangle
Left:	0
Bottom:	0
Width:	0.33000000000000002
Height:	2
Name:	R1

Figure 7.3: Object Dialog Window for geometry specification

Once the desired geometry is designed we choose Decompose Geometry form the initial / global window. This action leads to the decomposition of the initial geometry to subdomains.

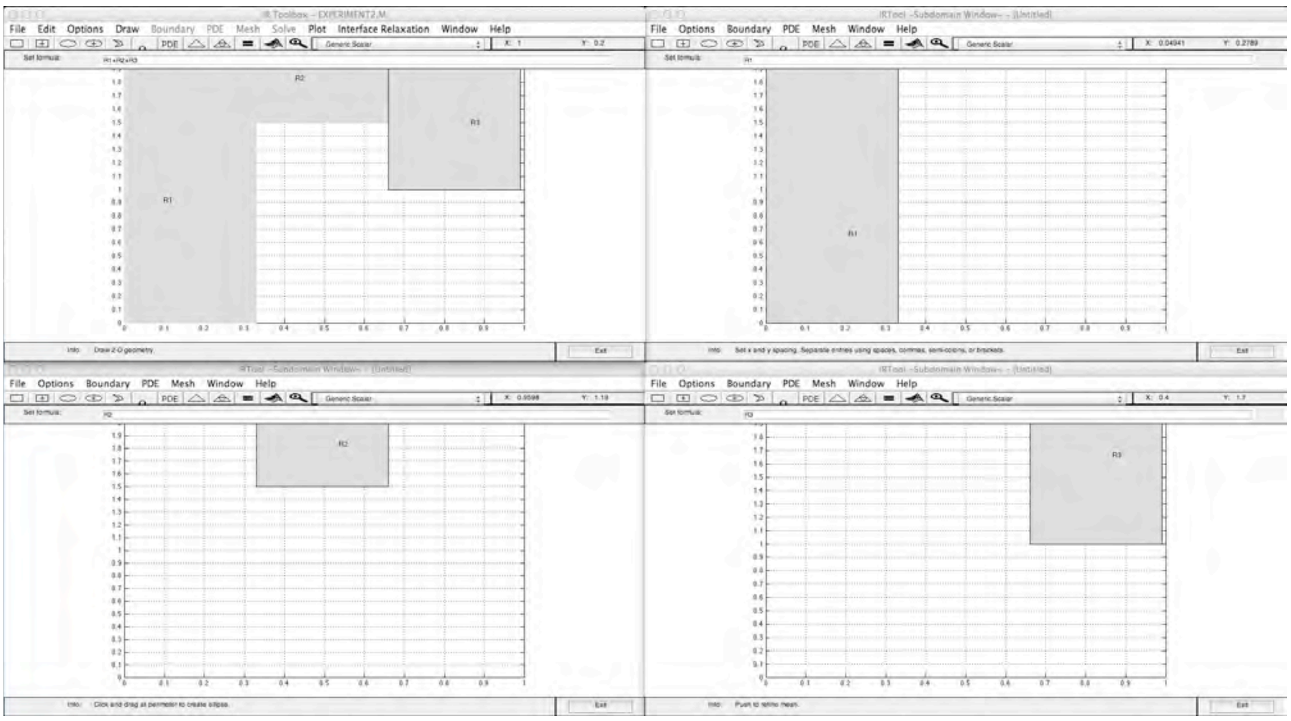

File Edit Options Draw Boundary PDE Mesh Solve Plot Interface Relaxation Window Help
R1
R2
R3
Draw 2-D geometry
Exit

Figure 7.4: Decomposition of the initial problem to subdomains

There after it is time to enter boundary mode for each subdomain, so we choose *Boundary Mode* from the Boundary Menu for each subdomain.

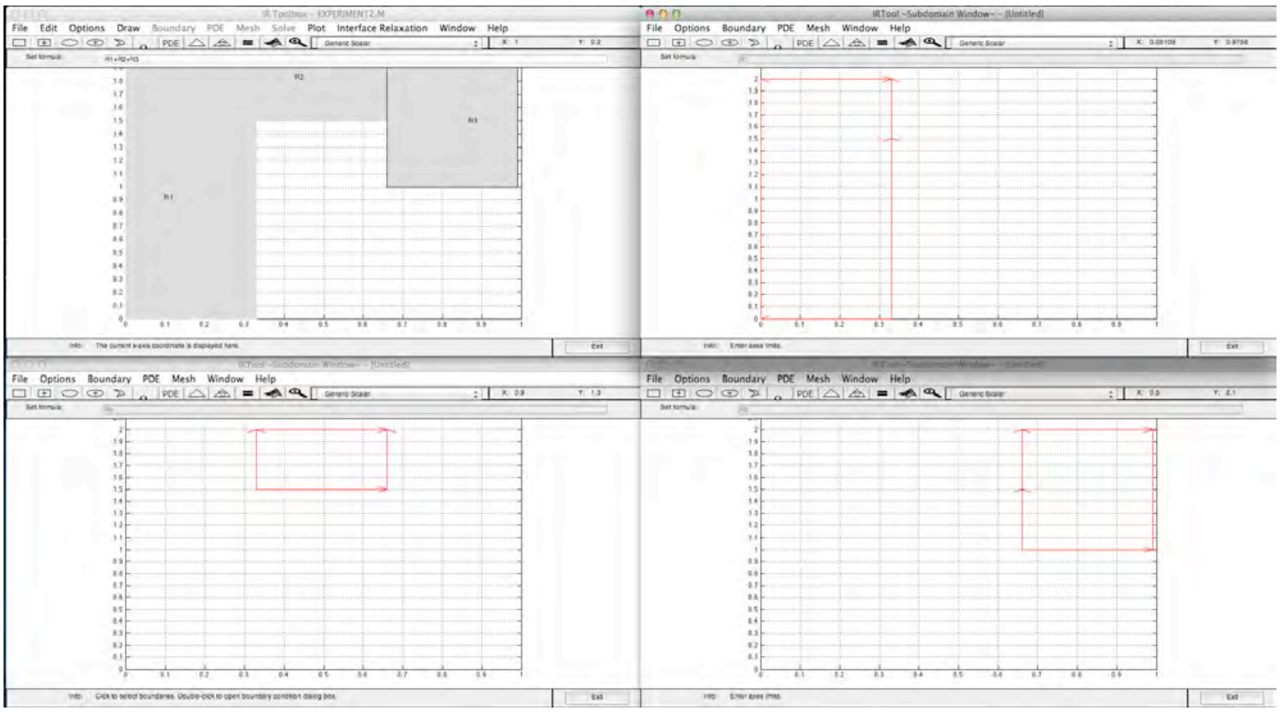
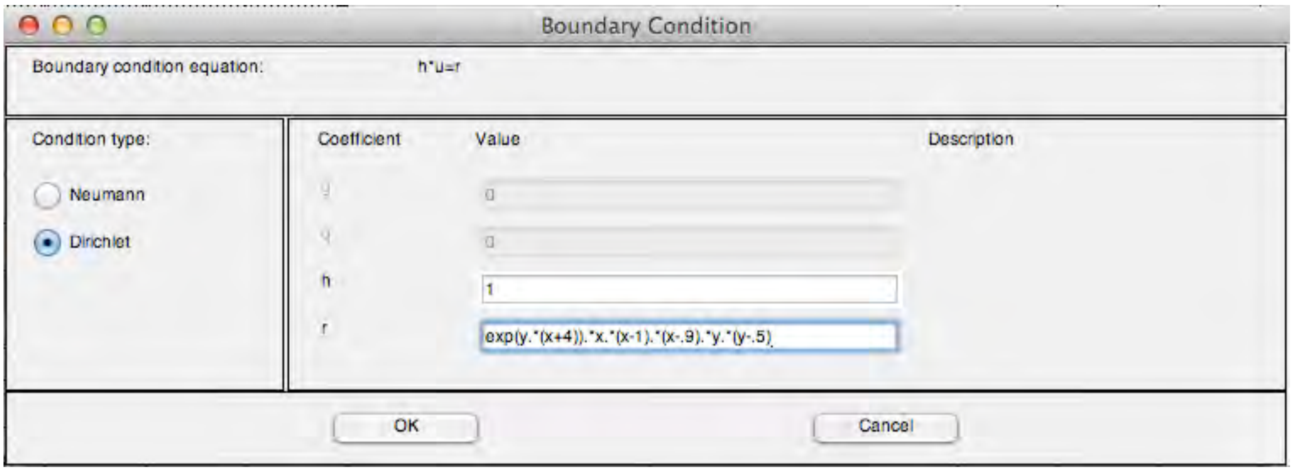

File Edit Options Draw Boundary PDE Mesh Solve Plot Interface Relaxation Window Help
R1
R2
R3
Exit

Figure 7.5: Boundary Mode of the subdomains

In *Boundary Mode* by double click on each boundary or interface we can give the input according to our problem. The two Figures below show the user input for both boundaries and interface according to the initial description of the problem.

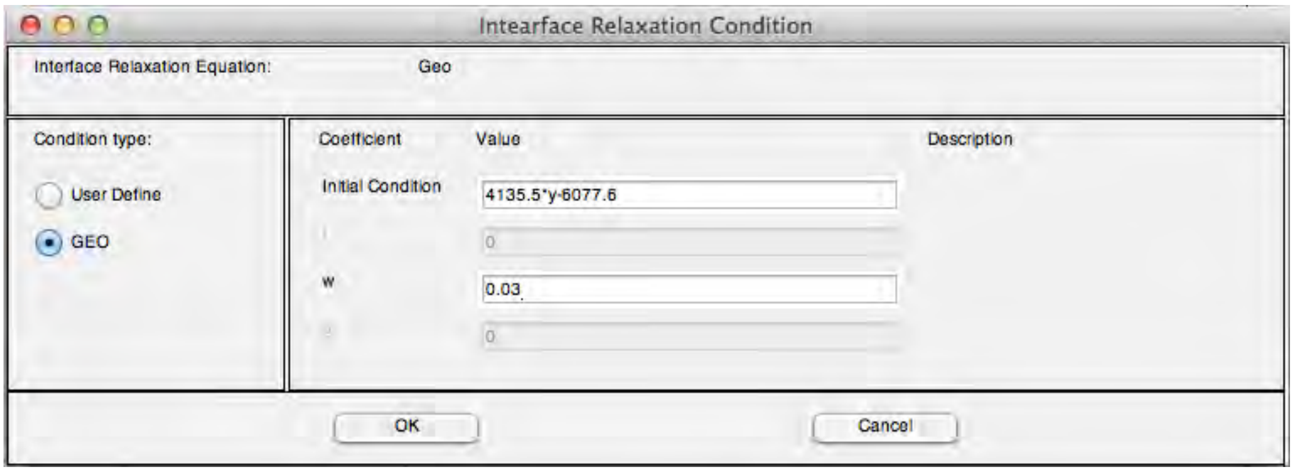

Boundary Condition

Boundary condition equation: h*u=r

Condition type: Neumann [✓ Dirichlet]

Coefficient	Value	Description
g	0	
q	0	
h	1	
r	exp(y.*(x+4)).*x.*(x-1).*(x-.9).*y.*(y-.5)	

Figure 7.6: Boundary Condition Window for Uniform Problem

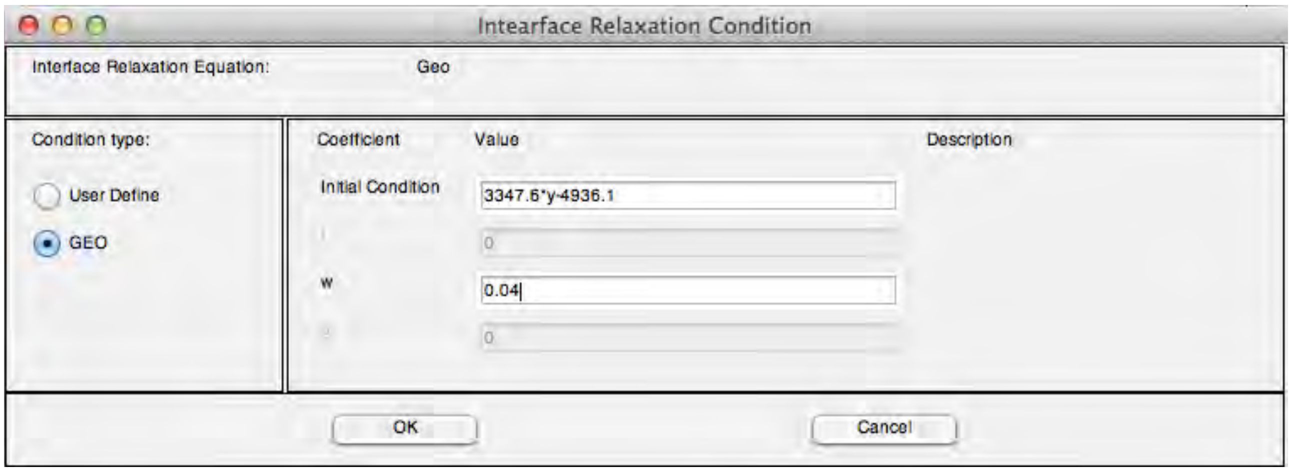

Intearface Relaxation Condition

Interface Relaxation Equation: Geo

Condition type: User Define [✓ GEO]

Coefficient	Value	Description
Initial Condition	4135.5*y-6077.6	
	0	
w	0.03	
	0	

Figure 7.7: Interface Relaxation Conditions Window for 1st Interface


Intearface Relaxation Condition

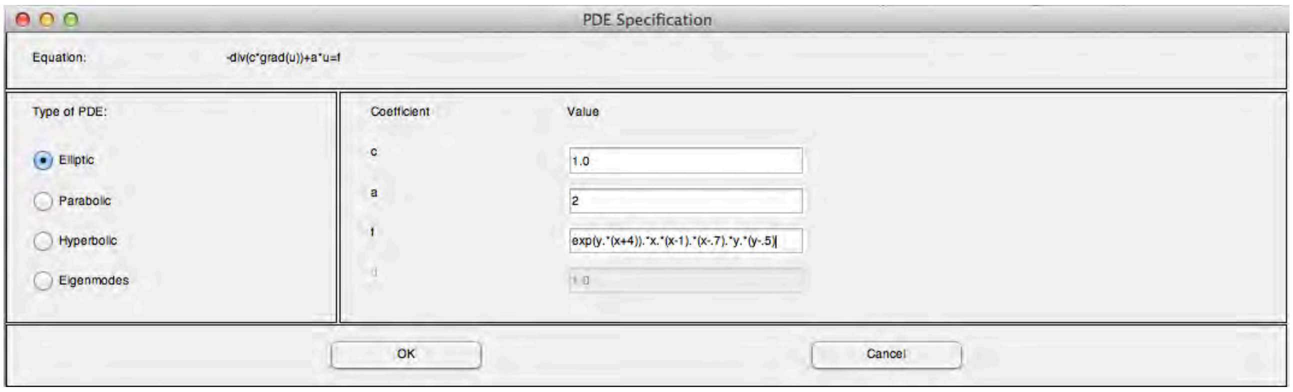
Interface Relaxation Equation: Geo

Condition type: User Define [✓ GEO]

Coefficient	Value	Description
Initial Condition	3347.6*y-4936.1	
	0	
w	0.04	
	0	

Figure 7.8 Interface Relaxations Conditions for 2nd Interface

Afterwards we insert *PDE Parameters* for each domain from the *PDE Menu*. At this stage *IRTool* has entered *PDE Mode*.


PDE Specification

Equation: -div(c*grad(u))+a*u=f

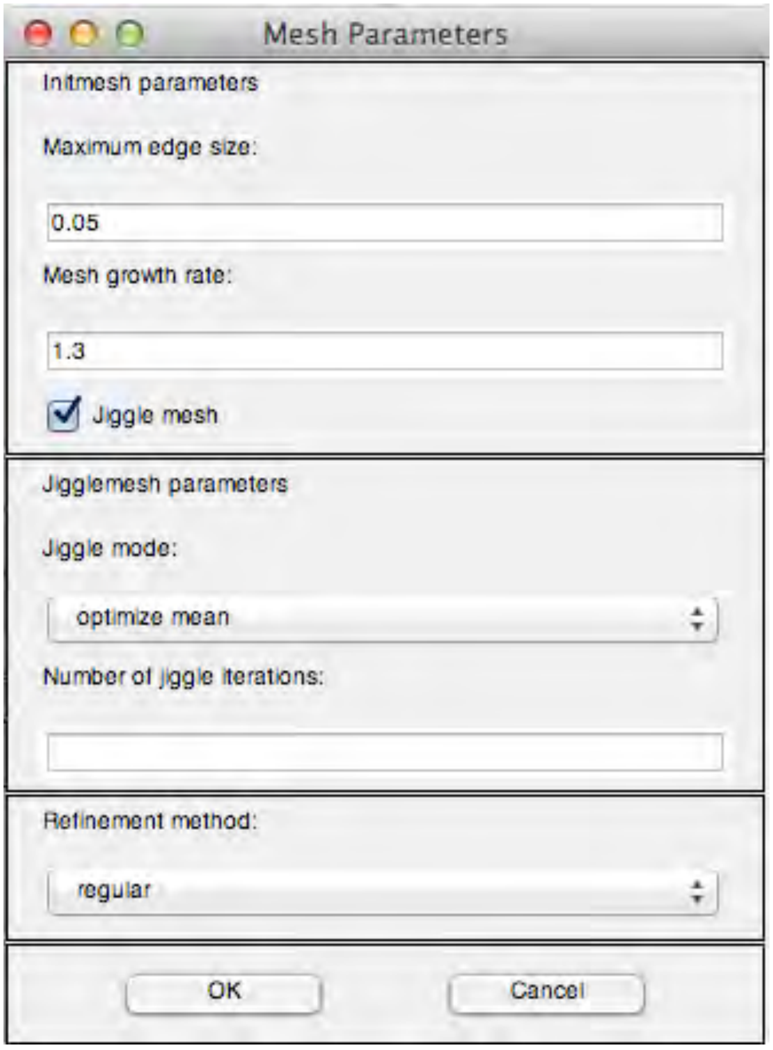
Type of PDE:

[✓ Elliptic] Parabolic Hyperbolic Eigenmodes

Coefficient	Value
c	1.0
a	2
f	exp(y.*(x+4)).*x.*(x-1).*(x-.7).*y.*(y-.5)
d	1.0

Figure 7.8: PDE Specification Window for Uniform Problem

Following the Menu from the main / global window we then must specify the hmax for the generation of the mesh in each subdomain. For our problem we use as input for mesh parameters as maximum edge size 0.05 (*hmax =0.05*) as shown in the Figure 7.9. .


Mesh Parameters

Initmesh parameters

Maximum edge size: 0.05

Mesh growth rate: 1.3

[✓ Jiggle mesh]

Jigglemesh parameters

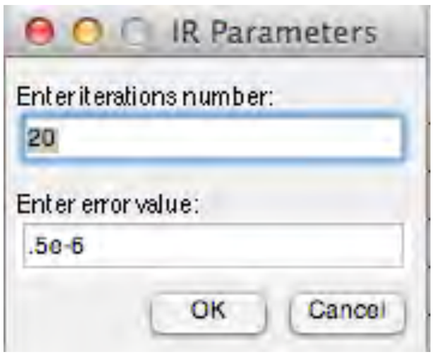
Jiggle mode: optimize mean

Number of jiggle iterations:

Refinement method: regular

Figure 7.9: Mesh Parameters Window for Uniform Problem

Before we proceed to the solution of our problem we have to specify the number of iterations and error value from the menu IR Parameters from the main / global window.



IR Parameters

Enter iterations number:

20

Enter error value:

.5e-6

Figure 7.10: IR Parameters Specification for uniform problem

The modeling of our problem is now complete. Lastly we have to choose *Solve MultiPDE* from the *Interface Relaxation Menu* in the main / global window. The Figure below shows the contour of the solution and the relaxation in each of the two interfaces.

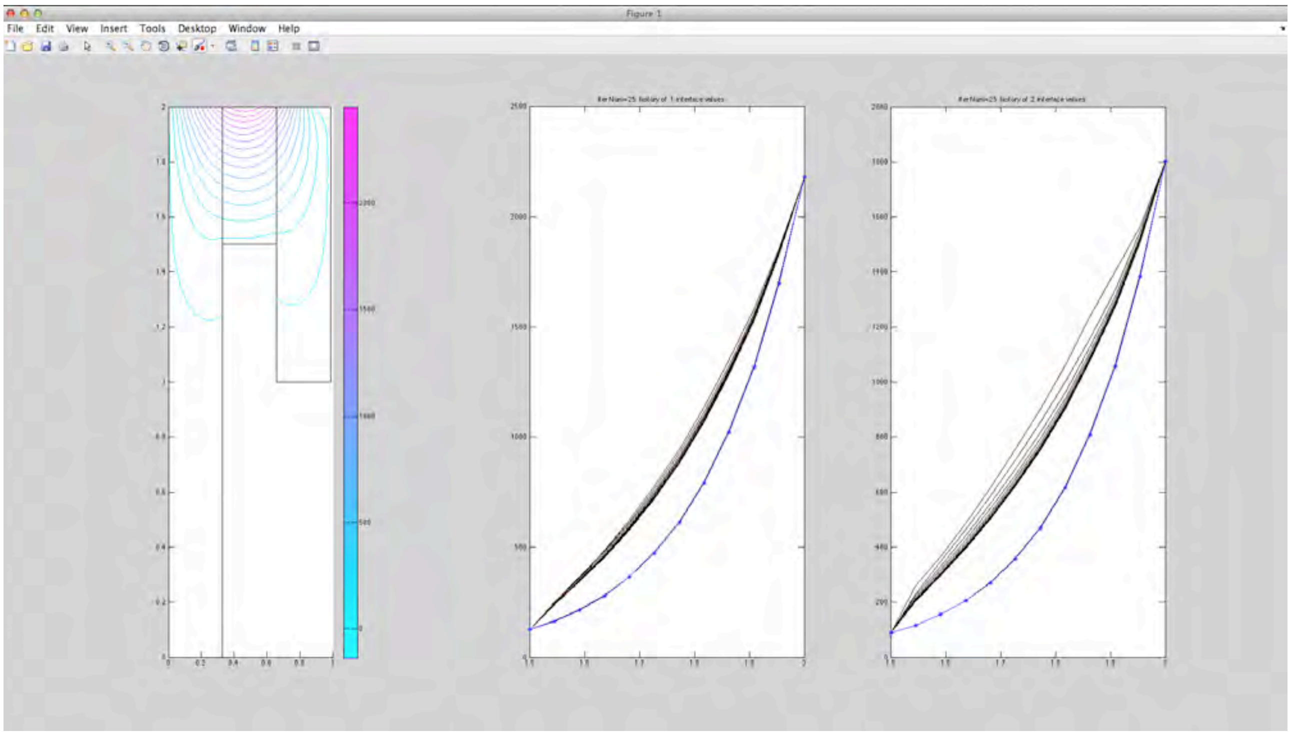


Figure 7.11: Contour plot for the solution and plots for the interface relaxation of the uniform problem during the 25th iteration

7.2.2 Non Uniform Problem - User Practice

The second PDE problem consists of a geometry that is decomposed into three domains with interfaces on $x_1 = 1/5$ and $x_2 = 1/2$. We use the same differential elliptic equation and true solution u(x,y), as we used in 7.2.1. For further practice the reader can model the problem him / herself. Below is the Figure of the solution.

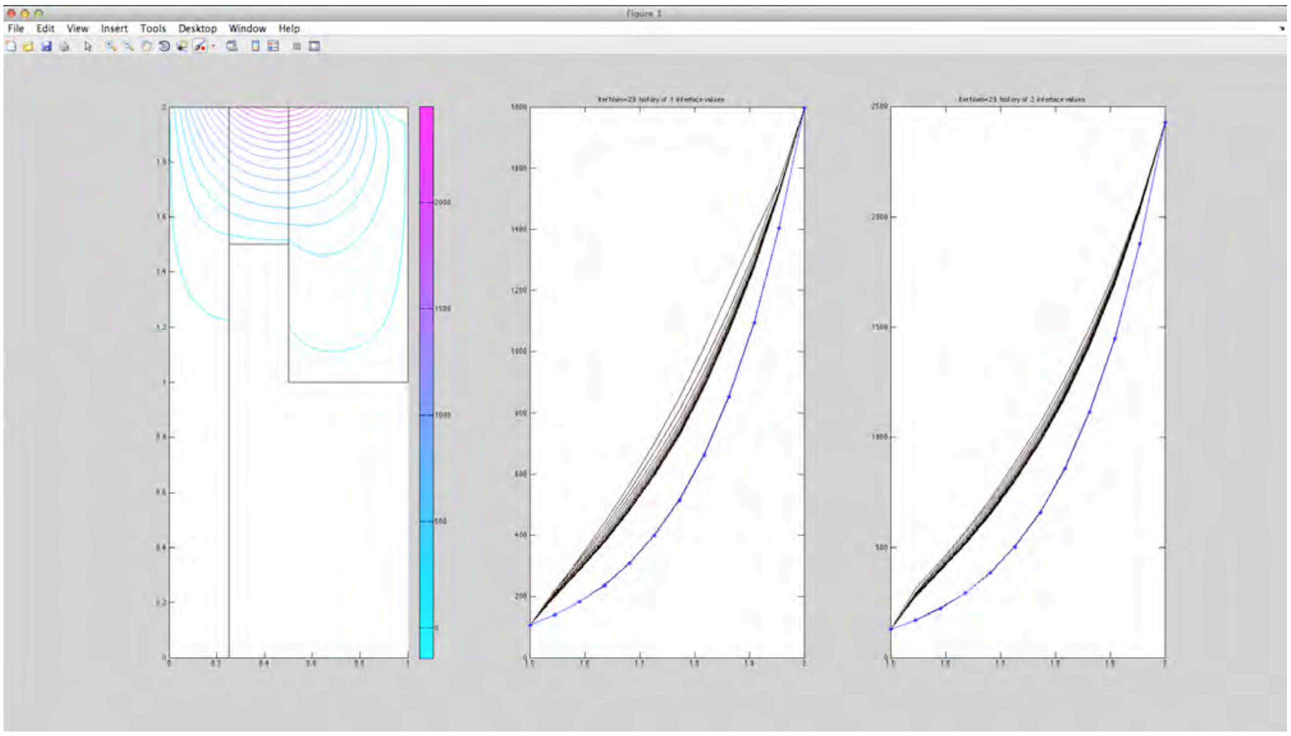


Figure 1
File Edit View Insert Tools Desktop Window Help
IterNum=29 history of 1 interface values
IterNum=29 history of 2 interface values

Figure 7.12: Contour plot for the solution and plots for the interface relaxation of the non uniform problem during the 29th iteration

Chapter 8: Conclusions

8.1 Conclusions

We propose an implementation of IRToolbox for composite PDE problems by properly combining existing models and software components. Our approach enjoys the following approaches:

Problem simplification. It dramatically simplifies the complexity of the physical problem by (1) considering subproblems that involve simpler local physical rules acting on simpler geometries, and (2) providing a convenient abstraction of the modeling and solution process while simultaneously providing a modeling practice that yields a closer representation of the physical world.

Reduction in software development time. It drastically reduces the time to develop a simulation engine by permitting the heavy reuse of legacy scientific software.

Numerical efficiency. It increases the efficiency of the overall numerical scheme by allowing one to reuse the most appropriate numerical method for each particular subproblem.

This thesis focuses on the implementation of IRToolbox that consists one proposal for solving MultiDomain / MultiPhysics Problems.

Bibliography

1] A. Quarteroni, A. Valli. Domain decomposition methods for partial differential equations. Oxford University Press, 2000.

[2] Panagiota Tsompanopoulou, Collaborative PDE Solvers: Theory and Practice. PhD Thesis, University of Crete, Department of Mathematics, 2000.

[3] P.E. Bjorstad, O. Widlund, To overlap or not overlap: A note on a domain decomposition method for elliptic problems (Pages: 1053 - 1061). 1989.

[4] PDE Toolbox, Matlab, http://www.mathworks.com/products/pde.

[5] H.S. McFaddin. An Object-based Problem Solving Environment for Collaborating PDE Solvers and Editors., PhD Thesis, Computer Science Department, Purdue University, 1992.

[6] T.T. Drashansky, An agent-based approach to building multidisciplinary problem solving environments. PhD Thesis, Computer Science Department, Purdue University, 1996.

[7] T.T. Drashansky, E.N. Houstis, N. Ramakrishnan, J.R. Rice, Networked Agents for Scientific Computing. ACM Communications, 1999.

[8] S.Karin, S.Graham, The high performance computing continuum., ACM Communications, 1998.

[9] T.Finin, Y.Labrou, J.Mayfield, KQML as an agent communication language. Cambridge, 1997.

[10] S.McFaddin, J.R. Rice, RELAX: A software platform for PDE Interface Relaxation methods.CSD-TR-91-018, 1991.

[11] J.R. Rice, P.Tsompanopoulou, E.Vavalis, Interface relaxation methods for elliptic differential equations. Applied Numerical Mathematics 32 (2000) 219-245, Purdue University, Computer Science Department, 2000.

[12] J.R. Rice, P.Tsompanopoulou, E.Vavalis, Fine Tuning interface relaxation methods for elliptic differential equations.Technical Report CSD-TR-98-017, Purdue University, Computer Science Department, 2002.

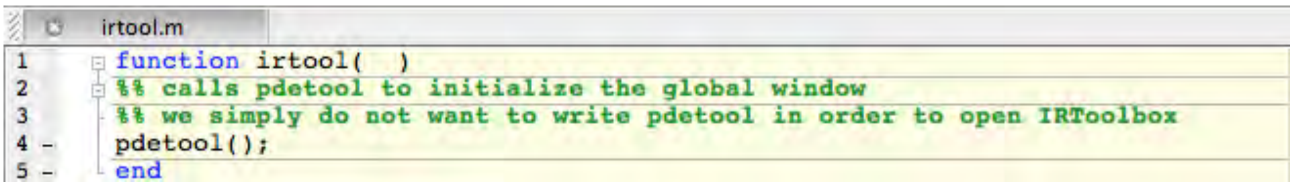
[13] Analysis of an Interface Relaxation Method for Composite Elliptic Differential Equations, P. Tsompanopoulou and E. Vavalis, Journal of Computational and Applied Mathematics, 226(2), (Apr. 2009), pp 370-387.

[14] An Experimental Study of Interface Relaxation Methods for Composite Elliptic Differential Equations, P. Tsompanopoulou and E. Vavalis, Applied Mathematical Modelling, 32, (Aug. 2008), pp 1620-1641.

Appendix

Initialization

1. irtool.m

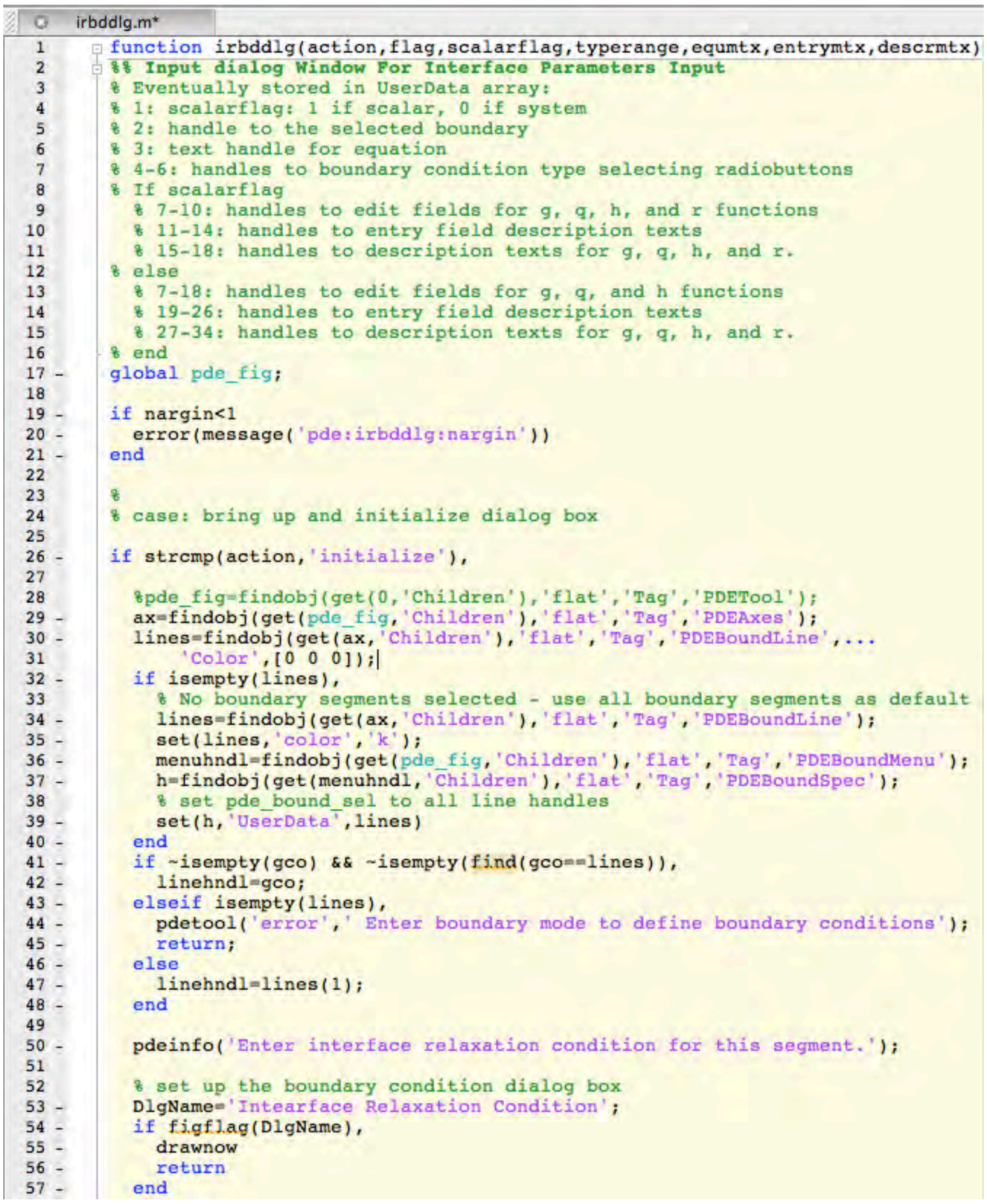


irtool.m

```matlab
function irtool(   )
%% calls pdetool to initialize the global window
%% we simply do not want to write pdetool in order to open IRToolbox
pdetool();
end
```

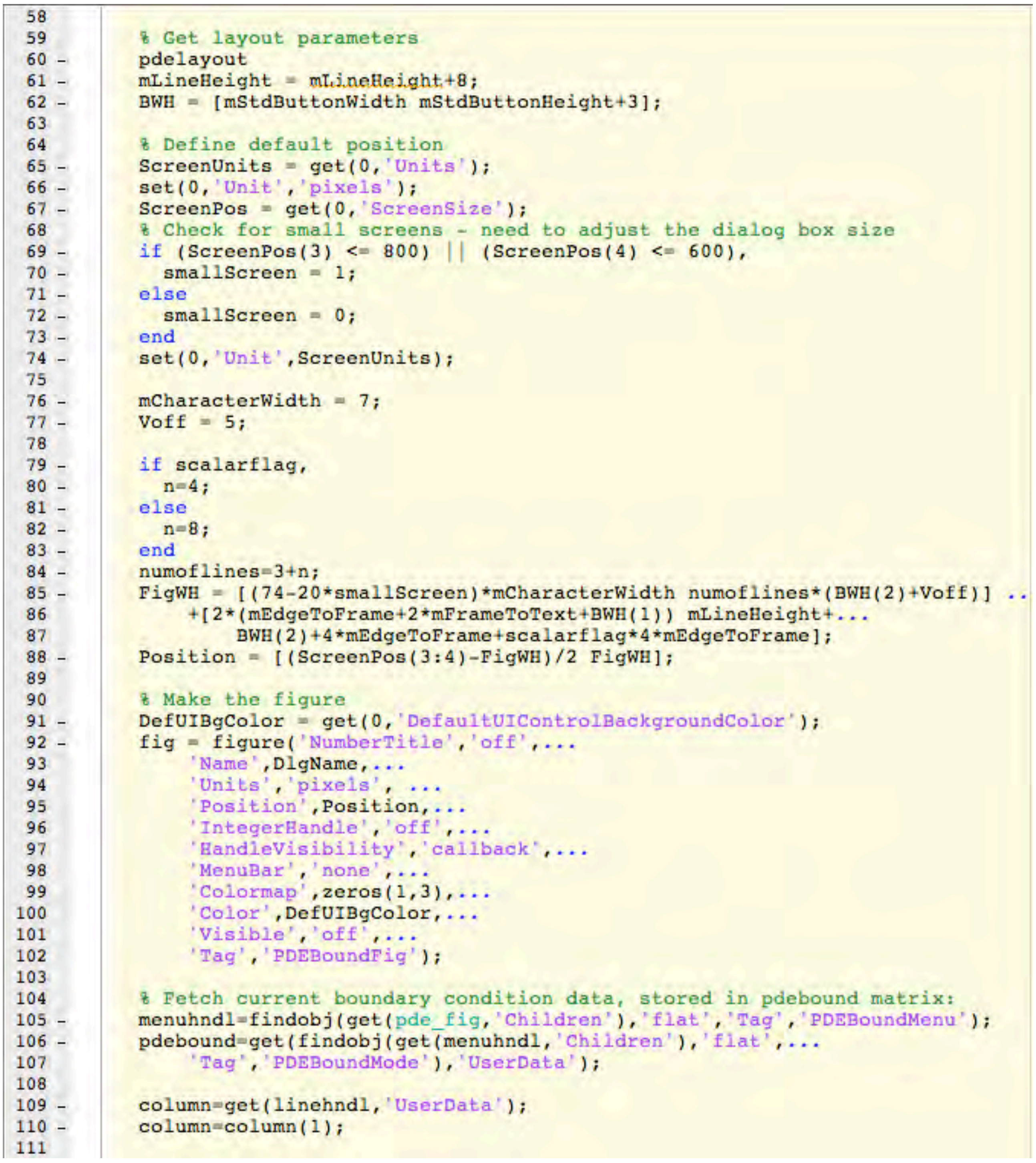
Boundary

1.*irbddlg.m*

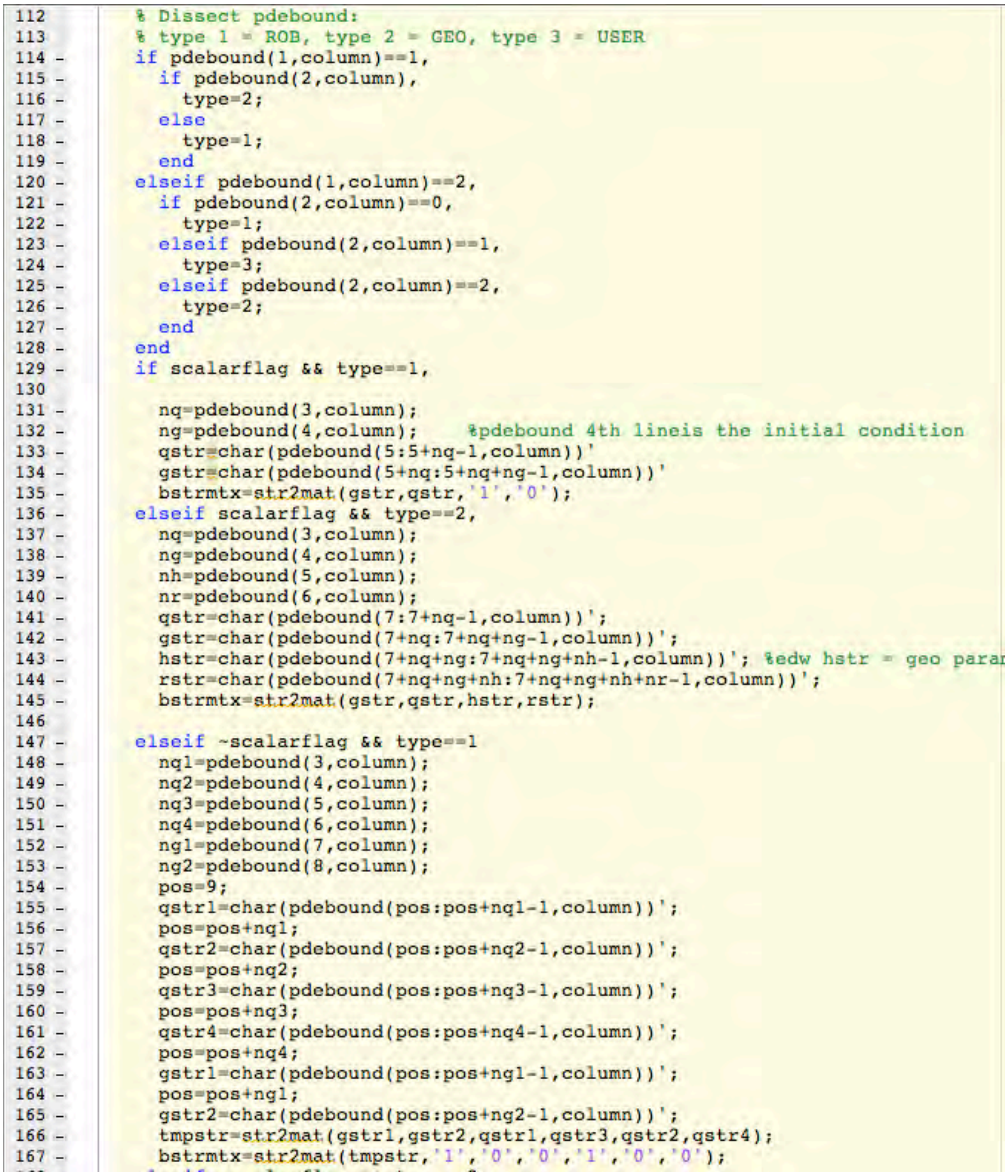


irbddlg.m*

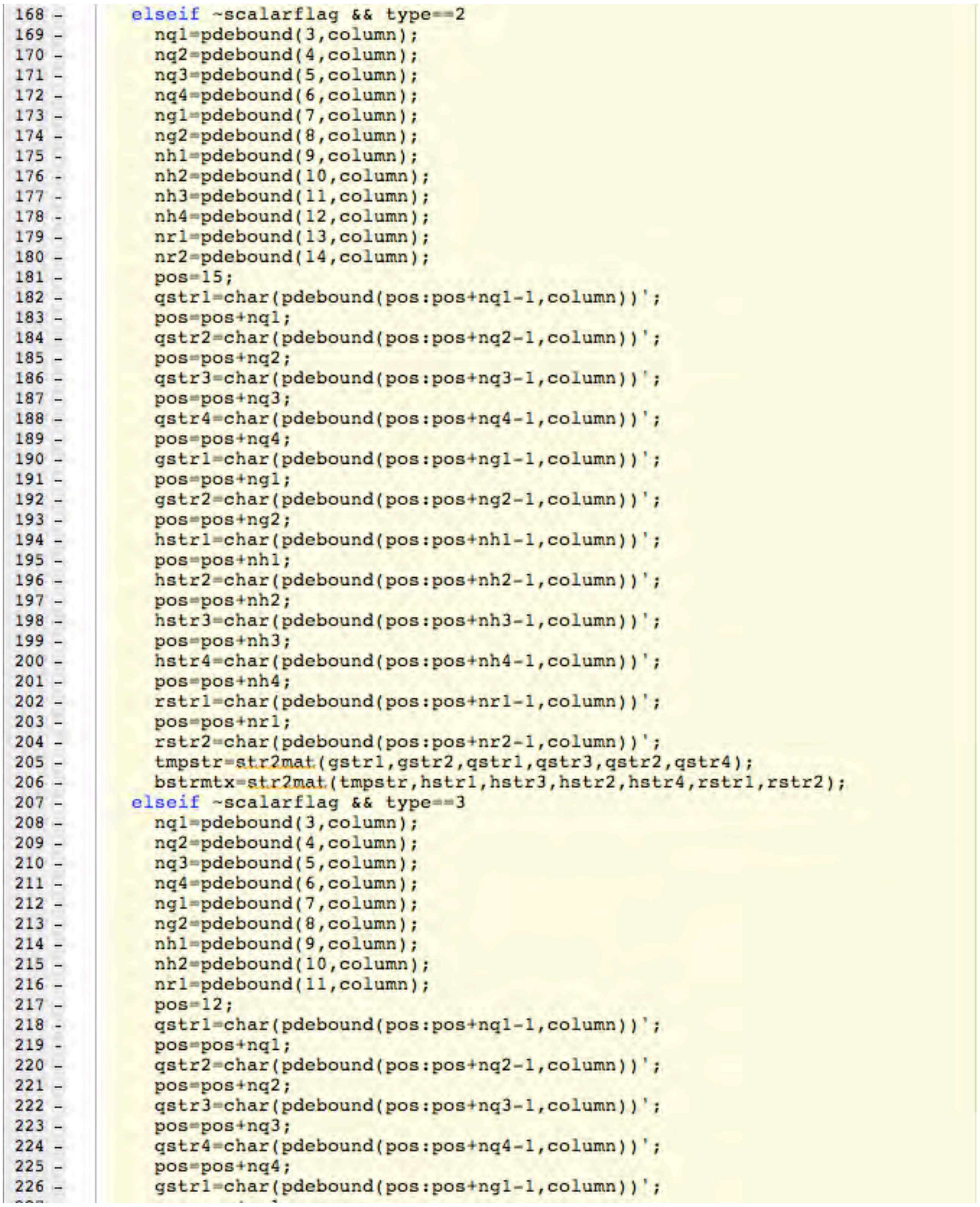
```matlab
function irbddlg(action,flag,scalarflag,typerange,equmtx,entrymtx,descrmtx)
%% Input dialog Window For Interface Parameters Input
% Eventually stored in UserData array:
% 1: scalarflag: 1 if scalar, 0 if system
% 2: handle to the selected boundary
% 3: text handle for equation
% 4-6: handles to boundary condition type selecting radiobuttons
% If scalarflag
  % 7-10: handles to edit fields for g, q, h, and r functions
  % 11-14: handles to entry field description texts
  % 15-18: handles to description texts for g, q, h, and r.
% else
  % 7-18: handles to edit fields for g, q, and h functions
  % 19-26: handles to entry field description texts
  % 27-34: handles to description texts for g, q, h, and r.
% end
global pde_fig;

if nargin<1
  error(message('pde:irbddlg:nargin'))
end

%
% case: bring up and initialize dialog box

if strcmp(action,'initialize'),

  %pde_fig=findobj(get(0,'Children'),'flat','Tag','PDETool');
  ax=findobj(get(pde_fig,'Children'),'flat','Tag','PDEAxes');
  lines=findobj(get(ax,'Children'),'flat','Tag','PDEBoundLine',...
      'Color',[0 0 0]);
  if isempty(lines),
    % No boundary segments selected - use all boundary segments as default
    lines=findobj(get(ax,'Children'),'flat','Tag','PDEBoundLine');
    set(lines,'color','k');
    menuhndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEBoundMenu');
    h=findobj(get(menuhndl,'Children'),'flat','Tag','PDEBoundSpec');
    % set pde_bound_sel to all line handles
    set(h,'UserData',lines)
  end
  if ~isempty(gco) && ~isempty(find(gco==lines)),
    linehndl=gco;
  elseif isempty(lines),
    pdetool('error',' Enter boundary mode to define boundary conditions');
    return;
  else
    linehndl=lines(1);
  end

  pdeinfo('Enter interface relaxation condition for this segment.');

  % set up the boundary condition dialog box
  DlgName='Intearface Relaxation Condition';
  if figflag(DlgName),
    drawnow
    return
  end
```



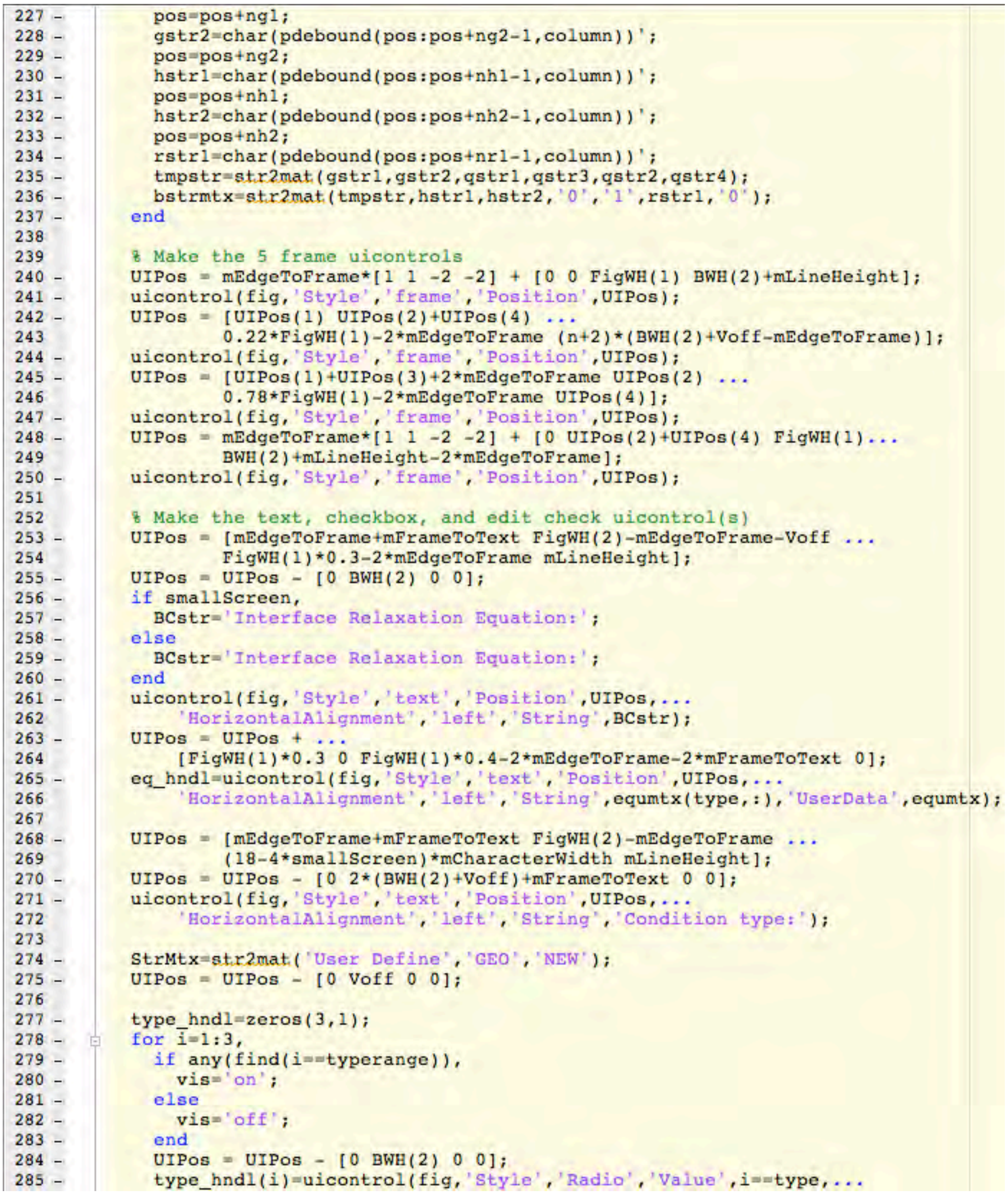
```
% Get layout parameters
pdelayout
mLineHeight = mLineHeight+8;
BWH = [mStdButtonWidth mStdButtonHeight+3];

% Define default position
ScreenUnits = get(0,'Units');
set(0,'Unit','pixels');
ScreenPos = get(0,'ScreenSize');
% Check for small screens - need to adjust the dialog box size
if (ScreenPos(3) <= 800) || (ScreenPos(4) <= 600),
  smallScreen = 1;
else
  smallScreen = 0;
end
set(0,'Unit',ScreenUnits);

mCharacterWidth = 7;
Voff = 5;

if scalarflag,
  n=4;
else
  n=8;
end
numoflines=3+n;
FigWH = [(74-20*smallScreen)*mCharacterWidth numoflines*(BWH(2)+Voff)] ..
    +[2*(mEdgeToFrame+2*mFrameToText+BWH(1)) mLineHeight+...
        BWH(2)+4*mEdgeToFrame+scalarflag*4*mEdgeToFrame];
Position = [(ScreenPos(3:4)-FigWH)/2 FigWH];

% Make the figure
DefUIBgColor = get(0,'DefaultUIControlBackgroundColor');
fig = figure('NumberTitle','off',...
    'Name',DlgName,...
    'Units','pixels', ...
    'Position',Position,...
    'IntegerHandle','off',...
    'HandleVisibility','callback',...
    'MenuBar','none',...
    'Colormap',zeros(1,3),...
    'Color',DefUIBgColor,...
    'Visible','off',...
    'Tag','PDEBoundFig');

% Fetch current boundary condition data, stored in pdebound matrix:
menuhndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEBoundMenu');
pdebound=get(findobj(get(menuhndl,'Children'),'flat',...
    'Tag','PDEBoundMode'),'UserData');

column=get(linehndl,'UserData');
column=column(1);
```



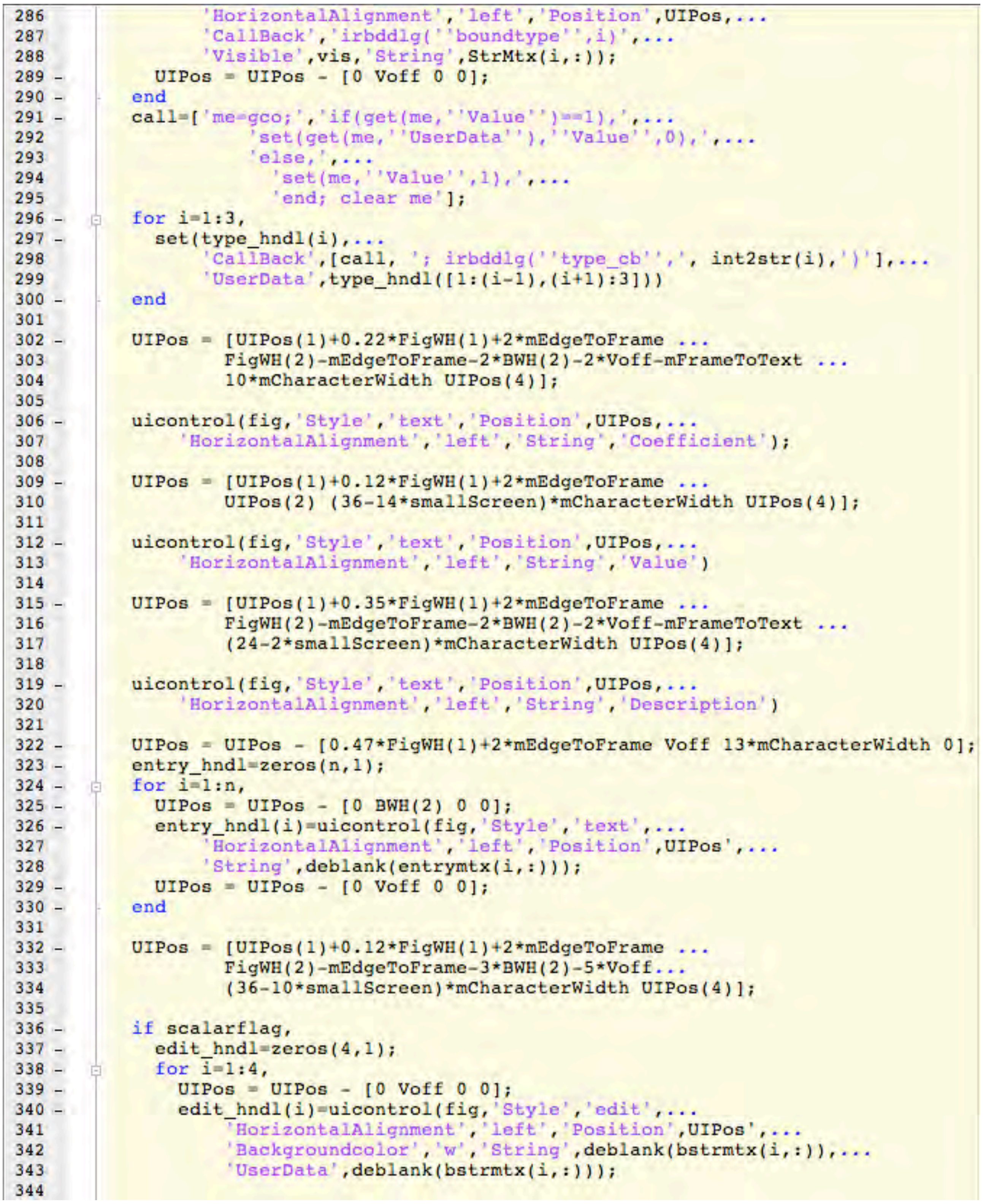
```
% Dissect pdebound:
% type 1 = ROB, type 2 = GEO, type 3 = USER
if pdebound(1,column)==1,
  if pdebound(2,column),
    type=2;
  else
    type=1;
  end
elseif pdebound(1,column)==2,
  if pdebound(2,column)==0,
    type=1;
  elseif pdebound(2,column)==1,
    type=3;
  elseif pdebound(2,column)==2,
    type=2;
  end
end
if scalarflag && type==1,

  nq=pdebound(3,column);
  ng=pdebound(4,column);    %pdebound 4th lineis the initial condition
  qstr=char(pdebound(5:5+nq-1,column))'
  gstr=char(pdebound(5+nq:5+nq+ng-1,column))'
  bstrmtx=str2mat(gstr,qstr,'1','0');
elseif scalarflag && type==2,
  nq=pdebound(3,column);
  ng=pdebound(4,column);
  nh=pdebound(5,column);
  nr=pdebound(6,column);
  qstr=char(pdebound(7:7+nq-1,column))';
  gstr=char(pdebound(7+nq:7+nq+ng-1,column))';
  hstr=char(pdebound(7+nq+ng:7+nq+ng+nh-1,column))'; %edw hstr = geo para
  rstr=char(pdebound(7+nq+ng+nh:7+nq+ng+nh+nr-1,column))';
  bstrmtx=str2mat(gstr,qstr,hstr,rstr);

elseif ~scalarflag && type==1
  nq1=pdebound(3,column);
  nq2=pdebound(4,column);
  nq3=pdebound(5,column);
  nq4=pdebound(6,column);
  ng1=pdebound(7,column);
  ng2=pdebound(8,column);
  pos=9;
  qstr1=char(pdebound(pos:pos+nq1-1,column))';
  pos=pos+nq1;
  qstr2=char(pdebound(pos:pos+nq2-1,column))';
  pos=pos+nq2;
  qstr3=char(pdebound(pos:pos+nq3-1,column))';
  pos=pos+nq3;
  qstr4=char(pdebound(pos:pos+nq4-1,column))';
  pos=pos+nq4;
  gstr1=char(pdebound(pos:pos+ng1-1,column))';
  pos=pos+ng1;
  gstr2=char(pdebound(pos:pos+ng2-1,column))';
  tmpstr=str2mat(gstr1,gstr2,qstr1,qstr3,qstr2,qstr4);
  bstrmtx=str2mat(tmpstr,'1','0','0','1','0','0');
```



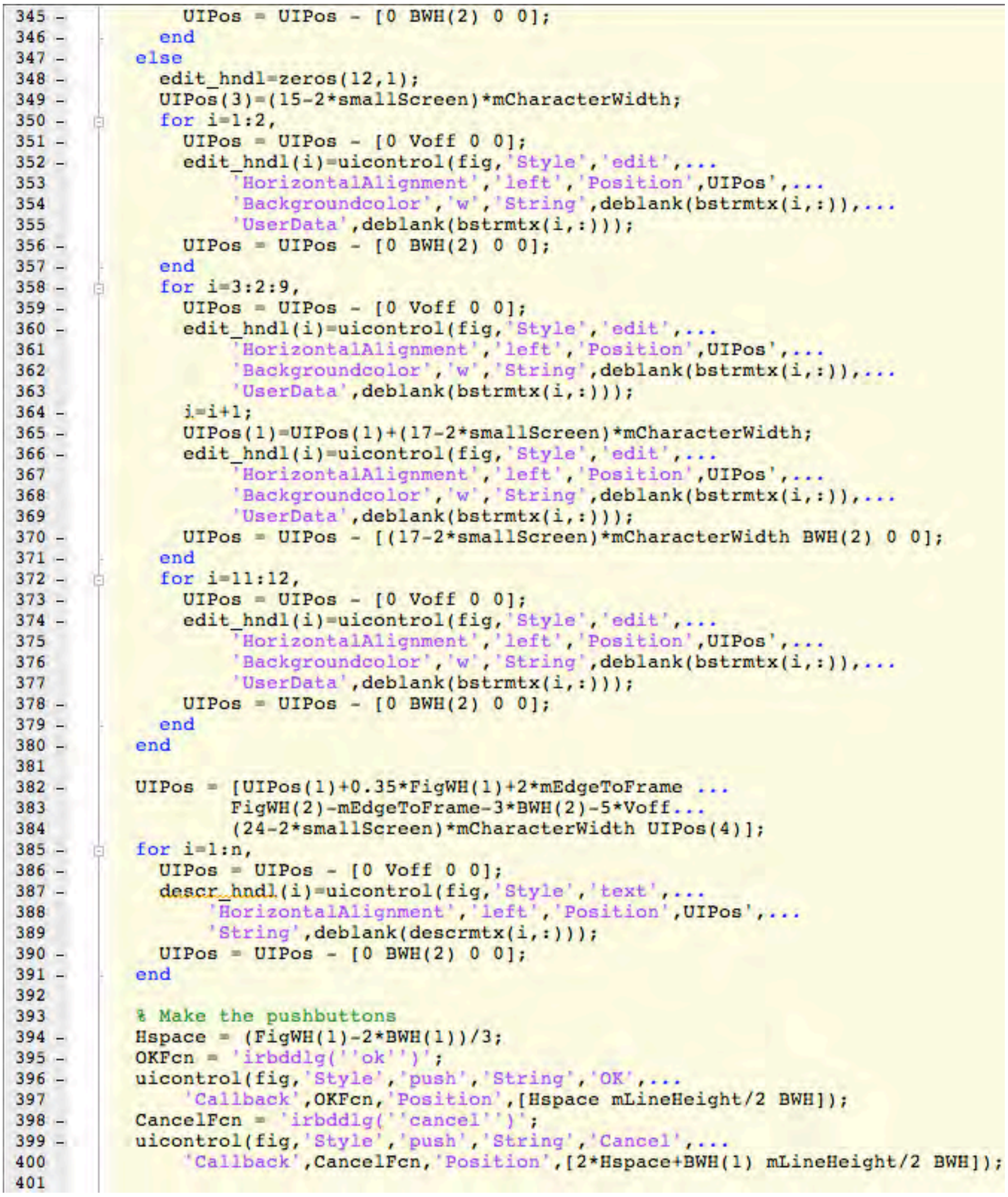
```
  elseif ~scalarflag && type==2
    nq1=pdebound(3,column);
    nq2=pdebound(4,column);
    nq3=pdebound(5,column);
    nq4=pdebound(6,column);
    ng1=pdebound(7,column);
    ng2=pdebound(8,column);
    nh1=pdebound(9,column);
    nh2=pdebound(10,column);
    nh3=pdebound(11,column);
    nh4=pdebound(12,column);
    nr1=pdebound(13,column);
    nr2=pdebound(14,column);
    pos=15;
    qstr1=char(pdebound(pos:pos+nq1-1,column))';
    pos=pos+nq1;
    qstr2=char(pdebound(pos:pos+nq2-1,column))';
    pos=pos+nq2;
    qstr3=char(pdebound(pos:pos+nq3-1,column))';
    pos=pos+nq3;
    qstr4=char(pdebound(pos:pos+nq4-1,column))';
    pos=pos+nq4;
    gstr1=char(pdebound(pos:pos+ng1-1,column))';
    pos=pos+ng1;
    gstr2=char(pdebound(pos:pos+ng2-1,column))';
    pos=pos+ng2;
    hstr1=char(pdebound(pos:pos+nh1-1,column))';
    pos=pos+nh1;
    hstr2=char(pdebound(pos:pos+nh2-1,column))';
    pos=pos+nh2;
    hstr3=char(pdebound(pos:pos+nh3-1,column))';
    pos=pos+nh3;
    hstr4=char(pdebound(pos:pos+nh4-1,column))';
    pos=pos+nh4;
    rstr1=char(pdebound(pos:pos+nr1-1,column))';
    pos=pos+nr1;
    rstr2=char(pdebound(pos:pos+nr2-1,column))';
    tmpstr=str2mat(gstr1,gstr2,qstr1,qstr3,qstr2,qstr4);
    bstrmtx=str2mat(tmpstr,hstr1,hstr3,hstr2,hstr4,rstr1,rstr2);
  elseif ~scalarflag && type==3
    nq1=pdebound(3,column);
    nq2=pdebound(4,column);
    nq3=pdebound(5,column);
    nq4=pdebound(6,column);
    ng1=pdebound(7,column);
    ng2=pdebound(8,column);
    nh1=pdebound(9,column);
    nh2=pdebound(10,column);
    nr1=pdebound(11,column);
    pos=12;
    qstr1=char(pdebound(pos:pos+nq1-1,column))';
    pos=pos+nq1;
    qstr2=char(pdebound(pos:pos+nq2-1,column))';
    pos=pos+nq2;
    qstr3=char(pdebound(pos:pos+nq3-1,column))';
    pos=pos+nq3;
    qstr4=char(pdebound(pos:pos+nq4-1,column))';
    pos=pos+nq4;
    gstr1=char(pdebound(pos:pos+ng1-1,column))';
```



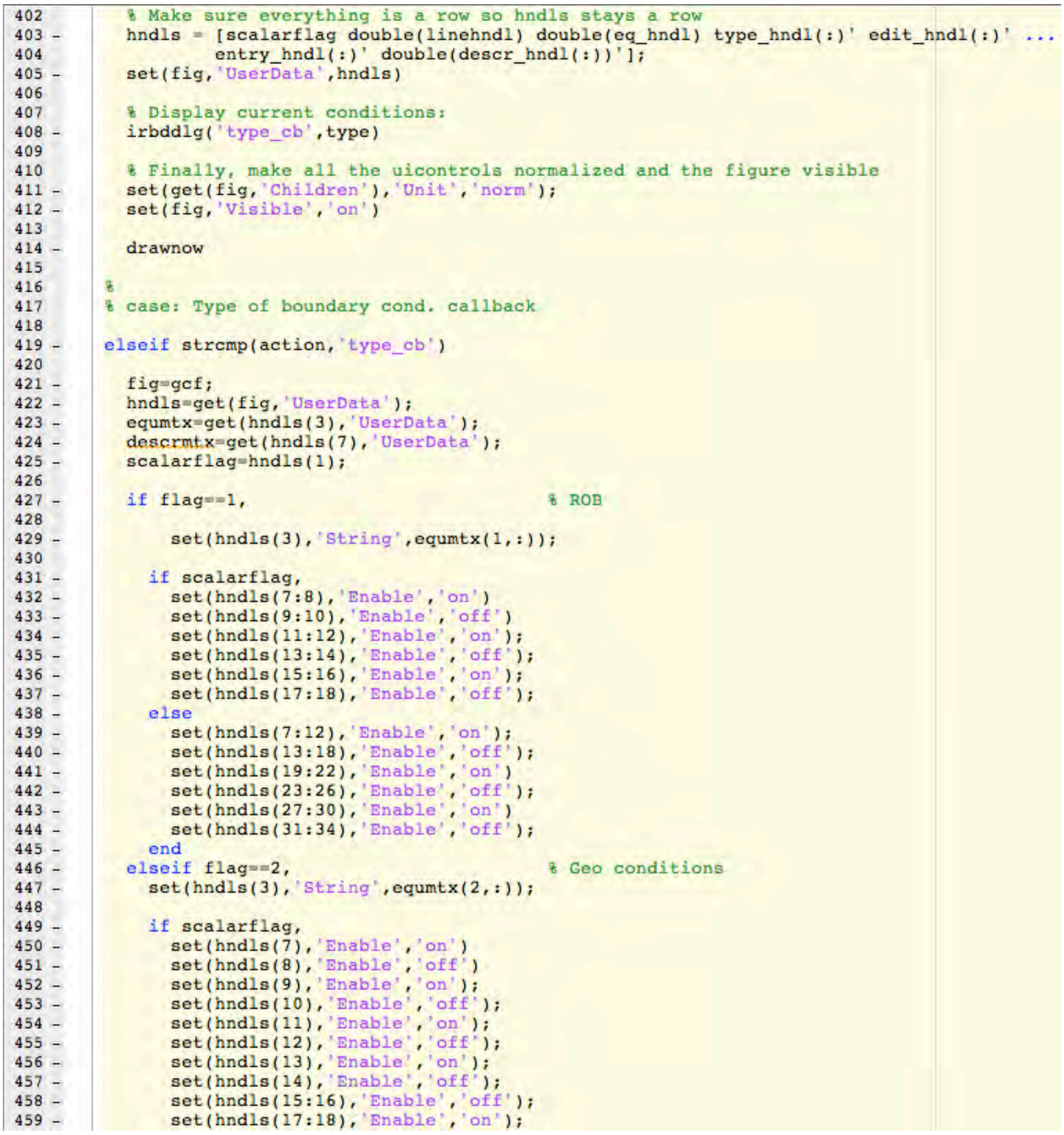
```
  pos=pos+ng1;
  gstr2=char(pdebound(pos:pos+ng2-1,column))';
  pos=pos+ng2;
  hstr1=char(pdebound(pos:pos+nh1-1,column))';
  pos=pos+nh1;
  hstr2=char(pdebound(pos:pos+nh2-1,column))';
  pos=pos+nh2;
  rstr1=char(pdebound(pos:pos+nr1-1,column))';
  tmpstr=str2mat(gstr1,gstr2,qstr1,qstr3,qstr2,qstr4);
  bstrmtx=str2mat(tmpstr,hstr1,hstr2,'0','1',rstr1,'0');
end

% Make the 5 frame uicontrols
UIPos = mEdgeToFrame*[1 1 -2 -2] + [0 0 FigWH(1) BWH(2)+mLineHeight];
uicontrol(fig,'Style','frame','Position',UIPos);
UIPos = [UIPos(1) UIPos(2)+UIPos(4) ...
        0.22*FigWH(1)-2*mEdgeToFrame (n+2)*(BWH(2)+Voff-mEdgeToFrame)];
uicontrol(fig,'Style','frame','Position',UIPos);
UIPos = [UIPos(1)+UIPos(3)+2*mEdgeToFrame UIPos(2) ...
        0.78*FigWH(1)-2*mEdgeToFrame UIPos(4)];
uicontrol(fig,'Style','frame','Position',UIPos);
UIPos = mEdgeToFrame*[1 1 -2 -2] + [0 UIPos(2)+UIPos(4) FigWH(1)...
        BWH(2)+mLineHeight-2*mEdgeToFrame];
uicontrol(fig,'Style','frame','Position',UIPos);

% Make the text, checkbox, and edit check uicontrol(s)
UIPos = [mEdgeToFrame+mFrameToText FigWH(2)-mEdgeToFrame-Voff ...
        FigWH(1)*0.3-2*mEdgeToFrame mLineHeight];
UIPos = UIPos - [0 BWH(2) 0 0];
if smallScreen,
  BCstr='Interface Relaxation Equation:';
else
  BCstr='Interface Relaxation Equation:';
end
uicontrol(fig,'Style','text','Position',UIPos,...
    'HorizontalAlignment','left','String',BCstr);
UIPos = UIPos + ...
    [FigWH(1)*0.3 0 FigWH(1)*0.4-2*mEdgeToFrame-2*mFrameToText 0];
eq_hndl=uicontrol(fig,'Style','text','Position',UIPos,...
    'HorizontalAlignment','left','String',equmtx(type,:),'UserData',equmtx);

UIPos = [mEdgeToFrame+mFrameToText FigWH(2)-mEdgeToFrame ...
        (18-4*smallScreen)*mCharacterWidth mLineHeight];
UIPos = UIPos - [0 2*(BWH(2)+Voff)+mFrameToText 0 0];
uicontrol(fig,'Style','text','Position',UIPos,...
    'HorizontalAlignment','left','String','Condition type:');

StrMtx=str2mat('User Define','GEO','NEW');
UIPos = UIPos - [0 Voff 0 0];

type_hndl=zeros(3,1);
for i=1:3,
  if any(find(i==typerange)),
    vis='on';
  else
    vis='off';
  end
  UIPos = UIPos - [0 BWH(2) 0 0];
  type_hndl(i)=uicontrol(fig,'Style','Radio','Value',i==type,...
```



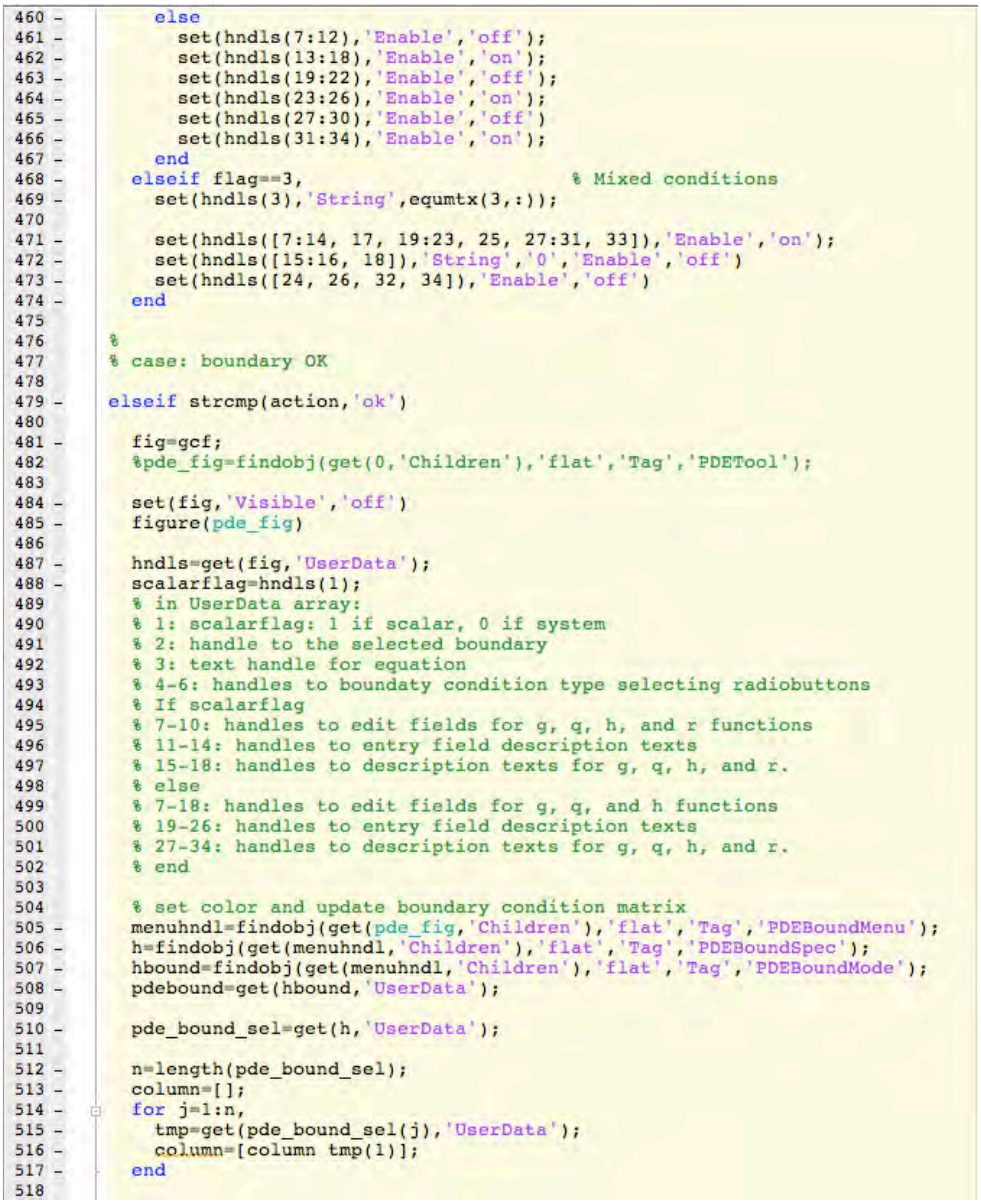
```
      'HorizontalAlignment','left','Position',UIPos,...
      'CallBack','irbddlg(''boundtype'',i)',...
      'Visible',vis,'String',StrMtx(i,:));
  UIPos = UIPos - [0 Voff 0 0];
end
call=['me=gco;','if(get(me,''Value'')==1),',...
          'set(get(me,''UserData''),''Value'',0),',...
          'else,',...
            'set(me,''Value'',1),',...
            'end; clear me'];
for i=1:3,
  set(type_hndl(i),...
      'CallBack',[call, '; irbddlg(''type_cb'',', int2str(i),')'],...
      'UserData',type_hndl([1:(i-1),(i+1):3]))
end

UIPos = [UIPos(1)+0.22*FigWH(1)+2*mEdgeToFrame ...
        FigWH(2)-mEdgeToFrame-2*BWH(2)-2*Voff-mFrameToText ...
        10*mCharacterWidth UIPos(4)];

uicontrol(fig,'Style','text','Position',UIPos,...
    'HorizontalAlignment','left','String','Coefficient');

UIPos = [UIPos(1)+0.12*FigWH(1)+2*mEdgeToFrame ...
        UIPos(2) (36-14*smallScreen)*mCharacterWidth UIPos(4)];

uicontrol(fig,'Style','text','Position',UIPos,...
    'HorizontalAlignment','left','String','Value')

UIPos = [UIPos(1)+0.35*FigWH(1)+2*mEdgeToFrame ...
        FigWH(2)-mEdgeToFrame-2*BWH(2)-2*Voff-mFrameToText ...
        (24-2*smallScreen)*mCharacterWidth UIPos(4)];

uicontrol(fig,'Style','text','Position',UIPos,...
    'HorizontalAlignment','left','String','Description')

UIPos = UIPos - [0.47*FigWH(1)+2*mEdgeToFrame Voff 13*mCharacterWidth 0];
entry_hndl=zeros(n,1);
for i=1:n,
  UIPos = UIPos - [0 BWH(2) 0 0];
  entry_hndl(i)=uicontrol(fig,'Style','text',...
      'HorizontalAlignment','left','Position',UIPos',...
      'String',deblank(entrymtx(i,:)));
  UIPos = UIPos - [0 Voff 0 0];
end

UIPos = [UIPos(1)+0.12*FigWH(1)+2*mEdgeToFrame ...
        FigWH(2)-mEdgeToFrame-3*BWH(2)-5*Voff...
        (36-10*smallScreen)*mCharacterWidth UIPos(4)];

if scalarflag,
  edit_hndl=zeros(4,1);
  for i=1:4,
    UIPos = UIPos - [0 Voff 0 0];
    edit_hndl(i)=uicontrol(fig,'Style','edit',...
        'HorizontalAlignment','left','Position',UIPos',...
        'Backgroundcolor','w','String',deblank(bstrmtx(i,:)),...
        'UserData',deblank(bstrmtx(i,:)));

```



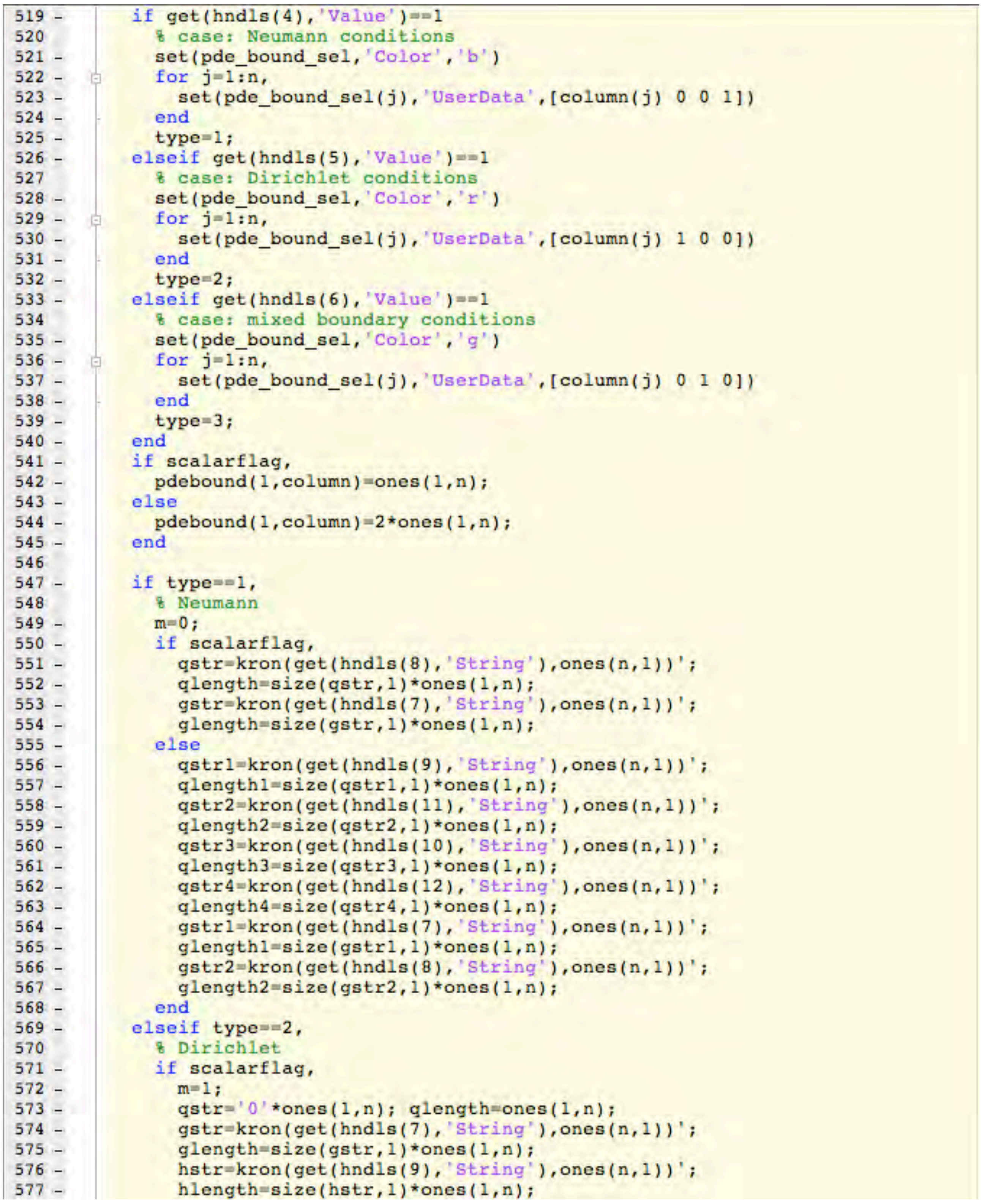
```
  UIPos = UIPos - [0 BWH(2) 0 0];
 end
else
 edit_hndl=zeros(12,1);
 UIPos(3)=(15-2*smallScreen)*mCharacterWidth;
 for i=1:2,
  UIPos = UIPos - [0 Voff 0 0];
  edit_hndl(i)=uicontrol(fig,'Style','edit',...
    'HorizontalAlignment','left','Position',UIPos',...
    'Backgroundcolor','w','String',deblank(bstrmtx(i,:)),...
    'UserData',deblank(bstrmtx(i,:)));
  UIPos = UIPos - [0 BWH(2) 0 0];
 end
 for i=3:2:9,
  UIPos = UIPos - [0 Voff 0 0];
  edit_hndl(i)=uicontrol(fig,'Style','edit',...
    'HorizontalAlignment','left','Position',UIPos',...
    'Backgroundcolor','w','String',deblank(bstrmtx(i,:)),...
    'UserData',deblank(bstrmtx(i,:)));
  i=i+1;
  UIPos(1)=UIPos(1)+(17-2*smallScreen)*mCharacterWidth;
  edit_hndl(i)=uicontrol(fig,'Style','edit',...
    'HorizontalAlignment','left','Position',UIPos',...
    'Backgroundcolor','w','String',deblank(bstrmtx(i,:)),...
    'UserData',deblank(bstrmtx(i,:)));
  UIPos = UIPos - [(17-2*smallScreen)*mCharacterWidth BWH(2) 0 0];
 end
 for i=11:12,
  UIPos = UIPos - [0 Voff 0 0];
  edit_hndl(i)=uicontrol(fig,'Style','edit',...
    'HorizontalAlignment','left','Position',UIPos',...
    'Backgroundcolor','w','String',deblank(bstrmtx(i,:)),...
    'UserData',deblank(bstrmtx(i,:)));
  UIPos = UIPos - [0 BWH(2) 0 0];
 end
end

UIPos = [UIPos(1)+0.35*FigWH(1)+2*mEdgeToFrame ...
    FigWH(2)-mEdgeToFrame-3*BWH(2)-5*Voff...
    (24-2*smallScreen)*mCharacterWidth UIPos(4)];
for i=1:n,
 UIPos = UIPos - [0 Voff 0 0];
 descr_hndl(i)=uicontrol(fig,'Style','text',...
   'HorizontalAlignment','left','Position',UIPos',...
   'String',deblank(descrmtx(i,:)));
 UIPos = UIPos - [0 BWH(2) 0 0];
end

% Make the pushbuttons
Hspace = (FigWH(1)-2*BWH(1))/3;
OKFcn = 'irbddlg(''ok'')';
uicontrol(fig,'Style','push','String','OK',...
  'Callback',OKFcn,'Position',[Hspace mLineHeight/2 BWH]);
CancelFcn = 'irbddlg(''cancel'')';
uicontrol(fig,'Style','push','String','Cancel',...
  'Callback',CancelFcn,'Position',[2*Hspace+BWH(1) mLineHeight/2 BWH]);
```



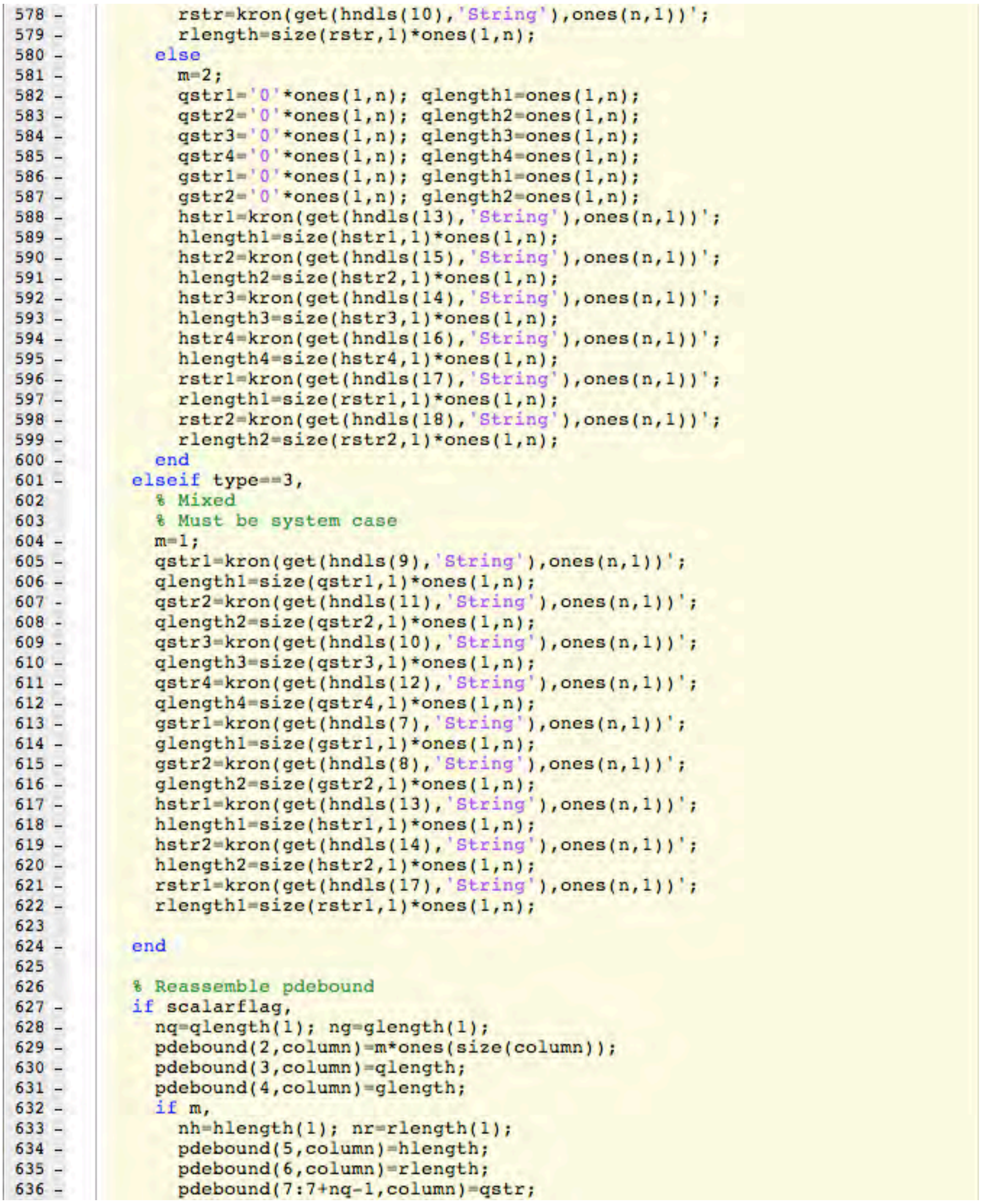
```
  % Make sure everything is a row so hndls stays a row
  hndls = [scalarflag double(linehndl) double(eq_hndl) type_hndl(:)' edit_hndl(:)' ...
          entry_hndl(:)' double(descr_hndl(:))'];
  set(fig,'UserData',hndls)

  % Display current conditions:
  irbddlg('type_cb',type)

  % Finally, make all the uicontrols normalized and the figure visible
  set(get(fig,'Children'),'Unit','norm');
  set(fig,'Visible','on')

  drawnow

%
% case: Type of boundary cond. callback

elseif strcmp(action,'type_cb')

  fig=gcf;
  hndls=get(fig,'UserData');
  equmtx=get(hndls(3),'UserData');
  descrmtx=get(hndls(7),'UserData');
  scalarflag=hndls(1);

  if flag==1,                           % ROB

      set(hndls(3),'String',equmtx(1,:));

    if scalarflag,
      set(hndls(7:8),'Enable','on')
      set(hndls(9:10),'Enable','off')
      set(hndls(11:12),'Enable','on');
      set(hndls(13:14),'Enable','off');
      set(hndls(15:16),'Enable','on');
      set(hndls(17:18),'Enable','off');
    else
      set(hndls(7:12),'Enable','on');
      set(hndls(13:18),'Enable','off');
      set(hndls(19:22),'Enable','on')
      set(hndls(23:26),'Enable','off');
      set(hndls(27:30),'Enable','on')
      set(hndls(31:34),'Enable','off');
    end
  elseif flag==2,                       % Geo conditions
    set(hndls(3),'String',equmtx(2,:));

    if scalarflag,
      set(hndls(7),'Enable','on')
      set(hndls(8),'Enable','off')
      set(hndls(9),'Enable','on');
      set(hndls(10),'Enable','off');
      set(hndls(11),'Enable','on');
      set(hndls(12),'Enable','off');
      set(hndls(13),'Enable','on');
      set(hndls(14),'Enable','off');
      set(hndls(15:16),'Enable','off');
      set(hndls(17:18),'Enable','on');
```



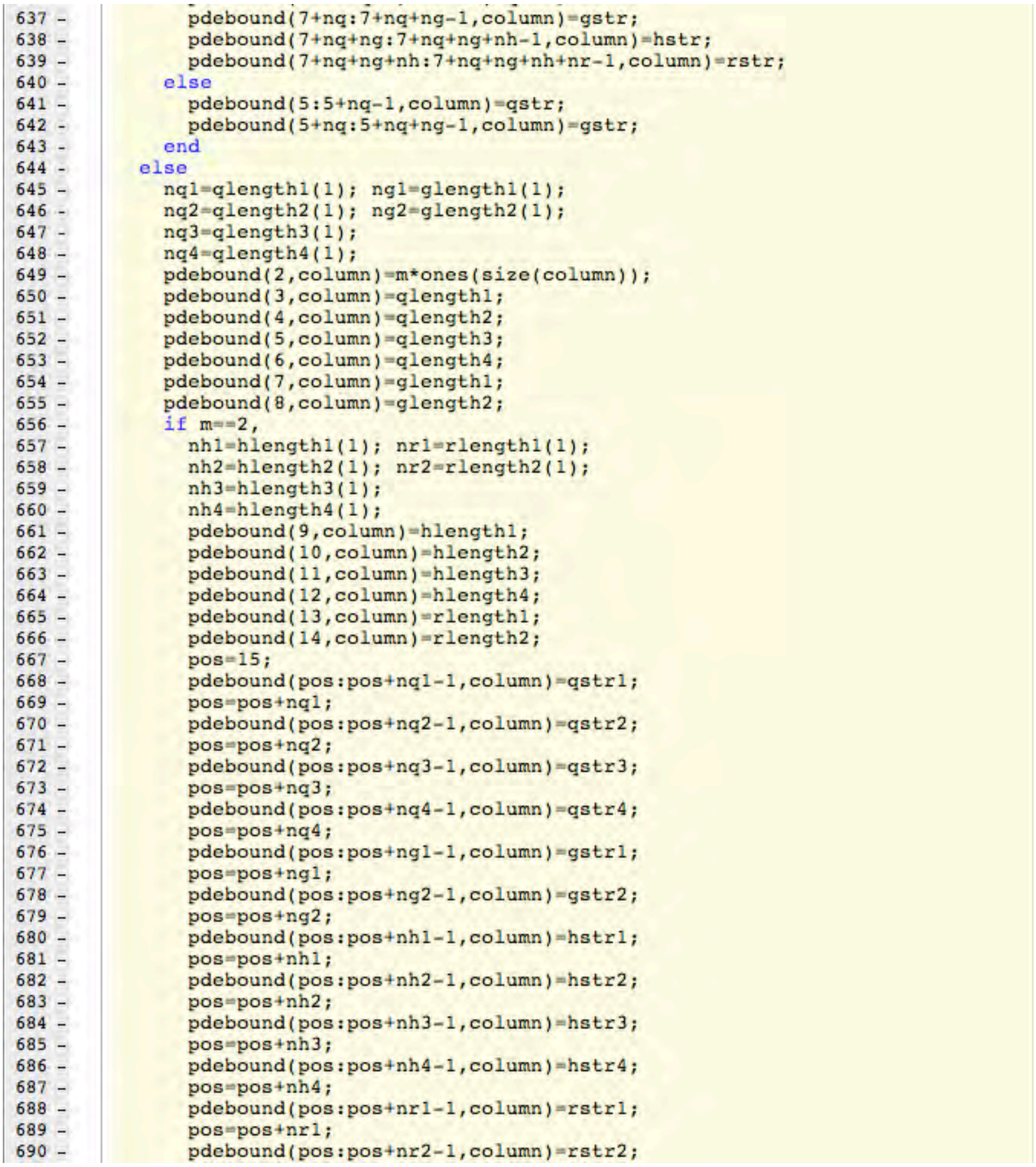
```
    else
      set(hndls(7:12),'Enable','off');
      set(hndls(13:18),'Enable','on');
      set(hndls(19:22),'Enable','off');
      set(hndls(23:26),'Enable','on');
      set(hndls(27:30),'Enable','off')
      set(hndls(31:34),'Enable','on');
    end
  elseif flag==3,                       % Mixed conditions
    set(hndls(3),'String',equmtx(3,:));

    set(hndls([7:14, 17, 19:23, 25, 27:31, 33]),'Enable','on');
    set(hndls([15:16, 18]),'String','0','Enable','off')
    set(hndls([24, 26, 32, 34]),'Enable','off')
  end

%
% case: boundary OK

elseif strcmp(action,'ok')

  fig=gcf;
  %pde_fig=findobj(get(0,'Children'),'flat','Tag','PDETool');

  set(fig,'Visible','off')
  figure(pde_fig)

  hndls=get(fig,'UserData');
  scalarflag=hndls(1);
  % in UserData array:
  % 1: scalarflag: 1 if scalar, 0 if system
  % 2: handle to the selected boundary
  % 3: text handle for equation
  % 4-6: handles to boundaty condition type selecting radiobuttons
  % If scalarflag
  % 7-10: handles to edit fields for g, q, h, and r functions
  % 11-14: handles to entry field description texts
  % 15-18: handles to description texts for g, q, h, and r.
  % else
  % 7-18: handles to edit fields for g, q, and h functions
  % 19-26: handles to entry field description texts
  % 27-34: handles to description texts for g, q, h, and r.
  % end

  % set color and update boundary condition matrix
  menuhndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEBoundMenu');
  h=findobj(get(menuhndl,'Children'),'flat','Tag','PDEBoundSpec');
  hbound=findobj(get(menuhndl,'Children'),'flat','Tag','PDEBoundMode');
  pdebound=get(hbound,'UserData');

  pde_bound_sel=get(h,'UserData');

  n=length(pde_bound_sel);
  column=[];
  for j=1:n,
    tmp=get(pde_bound_sel(j),'UserData');
    column=[column tmp(1)];
  end

```



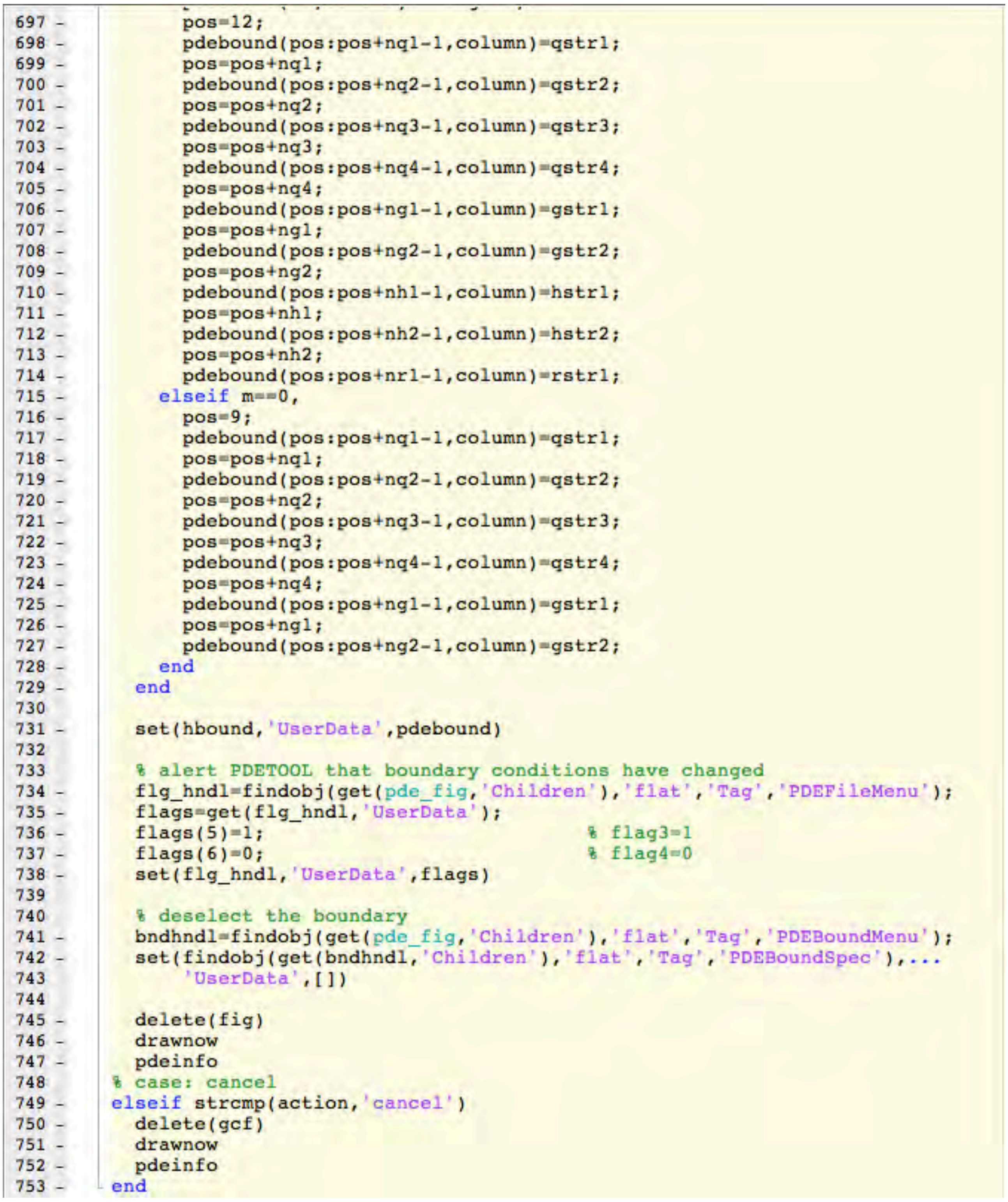
```
if get(hndls(4),'Value')==1
  % case: Neumann conditions
  set(pde_bound_sel,'Color','b')
  for j=1:n,
    set(pde_bound_sel(j),'UserData',[column(j) 0 0 1])
  end
  type=1;
elseif get(hndls(5),'Value')==1
  % case: Dirichlet conditions
  set(pde_bound_sel,'Color','r')
  for j=1:n,
    set(pde_bound_sel(j),'UserData',[column(j) 1 0 0])
  end
  type=2;
elseif get(hndls(6),'Value')==1
  % case: mixed boundary conditions
  set(pde_bound_sel,'Color','g')
  for j=1:n,
    set(pde_bound_sel(j),'UserData',[column(j) 0 1 0])
  end
  type=3;
end
if scalarflag,
  pdebound(1,column)=ones(1,n);
else
  pdebound(1,column)=2*ones(1,n);
end

if type==1,
  % Neumann
  m=0;
  if scalarflag,
    qstr=kron(get(hndls(8),'String'),ones(n,1))';
    qlength=size(qstr,1)*ones(1,n);
    gstr=kron(get(hndls(7),'String'),ones(n,1))';
    glength=size(gstr,1)*ones(1,n);
  else
    qstr1=kron(get(hndls(9),'String'),ones(n,1))';
    qlength1=size(qstr1,1)*ones(1,n);
    qstr2=kron(get(hndls(11),'String'),ones(n,1))';
    qlength2=size(qstr2,1)*ones(1,n);
    qstr3=kron(get(hndls(10),'String'),ones(n,1))';
    qlength3=size(qstr3,1)*ones(1,n);
    qstr4=kron(get(hndls(12),'String'),ones(n,1))';
    qlength4=size(qstr4,1)*ones(1,n);
    gstr1=kron(get(hndls(7),'String'),ones(n,1))';
    glength1=size(gstr1,1)*ones(1,n);
    gstr2=kron(get(hndls(8),'String'),ones(n,1))';
    glength2=size(gstr2,1)*ones(1,n);
  end
elseif type==2,
  % Dirichlet
  if scalarflag,
    m=1;
    qstr='0'*ones(1,n); qlength=ones(1,n);
    gstr=kron(get(hndls(7),'String'),ones(n,1))';
    glength=size(gstr,1)*ones(1,n);
    hstr=kron(get(hndls(9),'String'),ones(n,1))';
    hlength=size(hstr,1)*ones(1,n);
```



```
      rstr=kron(get(hndls(10),'String'),ones(n,1))';
      rlength=size(rstr,1)*ones(1,n);
    else
      m=2;
      qstr1='0'*ones(1,n); qlength1=ones(1,n);
      qstr2='0'*ones(1,n); qlength2=ones(1,n);
      qstr3='0'*ones(1,n); qlength3=ones(1,n);
      qstr4='0'*ones(1,n); qlength4=ones(1,n);
      gstr1='0'*ones(1,n); glength1=ones(1,n);
      gstr2='0'*ones(1,n); glength2=ones(1,n);
      hstr1=kron(get(hndls(13),'String'),ones(n,1))';
      hlength1=size(hstr1,1)*ones(1,n);
      hstr2=kron(get(hndls(15),'String'),ones(n,1))';
      hlength2=size(hstr2,1)*ones(1,n);
      hstr3=kron(get(hndls(14),'String'),ones(n,1))';
      hlength3=size(hstr3,1)*ones(1,n);
      hstr4=kron(get(hndls(16),'String'),ones(n,1))';
      hlength4=size(hstr4,1)*ones(1,n);
      rstr1=kron(get(hndls(17),'String'),ones(n,1))';
      rlength1=size(rstr1,1)*ones(1,n);
      rstr2=kron(get(hndls(18),'String'),ones(n,1))';
      rlength2=size(rstr2,1)*ones(1,n);
    end
  elseif type==3,
    % Mixed
    % Must be system case
    m=1;
    qstr1=kron(get(hndls(9),'String'),ones(n,1))';
    qlength1=size(qstr1,1)*ones(1,n);
    qstr2=kron(get(hndls(11),'String'),ones(n,1))';
    qlength2=size(qstr2,1)*ones(1,n);
    qstr3=kron(get(hndls(10),'String'),ones(n,1))';
    qlength3=size(qstr3,1)*ones(1,n);
    qstr4=kron(get(hndls(12),'String'),ones(n,1))';
    qlength4=size(qstr4,1)*ones(1,n);
    gstr1=kron(get(hndls(7),'String'),ones(n,1))';
    glength1=size(gstr1,1)*ones(1,n);
    gstr2=kron(get(hndls(8),'String'),ones(n,1))';
    glength2=size(gstr2,1)*ones(1,n);
    hstr1=kron(get(hndls(13),'String'),ones(n,1))';
    hlength1=size(hstr1,1)*ones(1,n);
    hstr2=kron(get(hndls(14),'String'),ones(n,1))';
    hlength2=size(hstr2,1)*ones(1,n);
    rstr1=kron(get(hndls(17),'String'),ones(n,1))';
    rlength1=size(rstr1,1)*ones(1,n);

  end

  % Reassemble pdebound
  if scalarflag,
    nq=qlength(1); ng=glength(1);
    pdebound(2,column)=m*ones(size(column));
    pdebound(3,column)=qlength;
    pdebound(4,column)=glength;
    if m,
      nh=hlength(1); nr=rlength(1);
      pdebound(5,column)=hlength;
      pdebound(6,column)=rlength;
      pdebound(7:7+nq-1,column)=qstr;
```



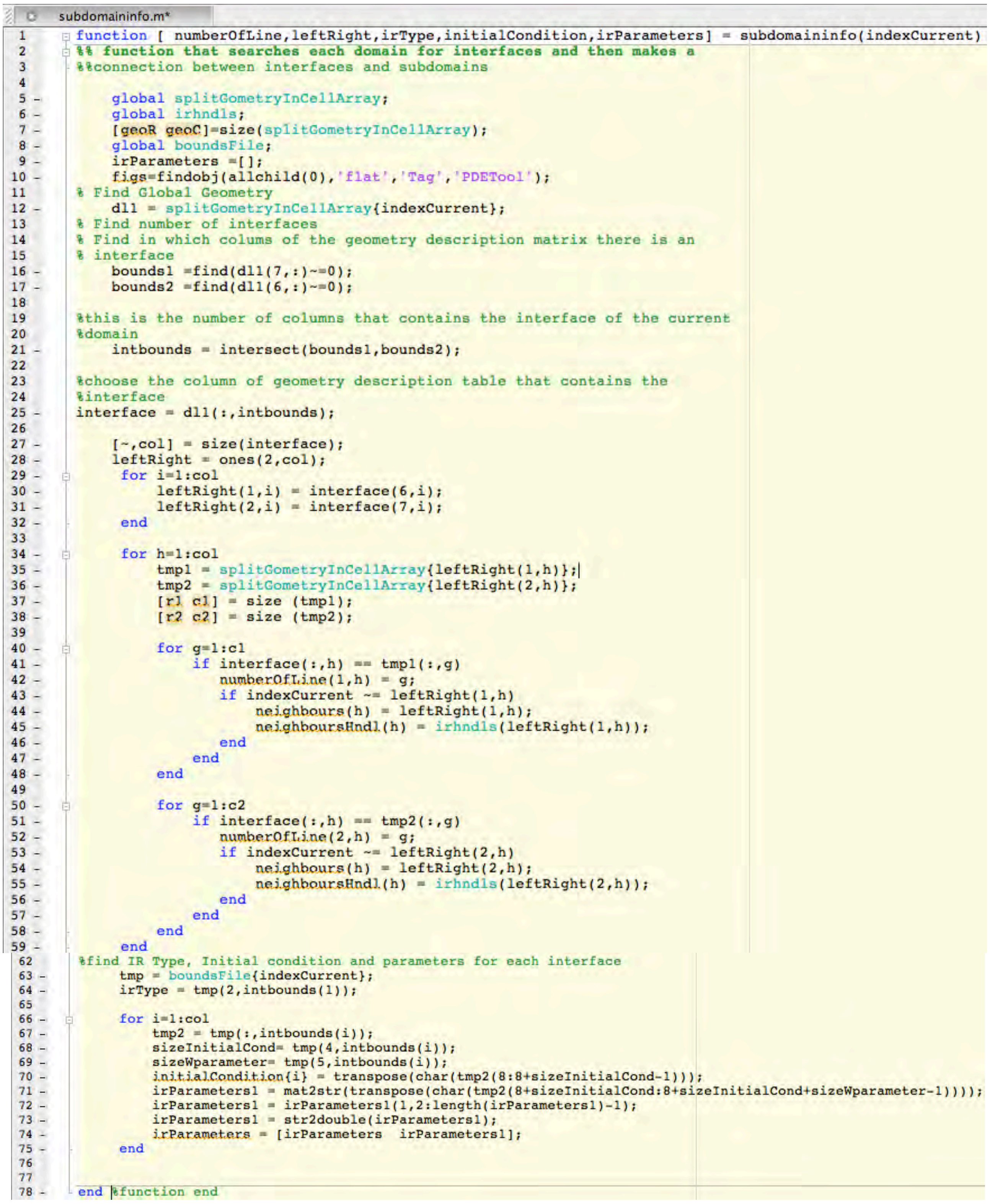
```
        pdebound(7+nq:7+nq+ng-1,column)=gstr;
        pdebound(7+nq+ng:7+nq+ng+nh-1,column)=hstr;
        pdebound(7+nq+ng+nh:7+nq+ng+nh+nr-1,column)=rstr;
      else
        pdebound(5:5+nq-1,column)=qstr;
        pdebound(5+nq:5+nq+ng-1,column)=gstr;
      end
    else
      nq1=qlength1(1); ng1=glength1(1);
      nq2=qlength2(1); ng2=glength2(1);
      nq3=qlength3(1);
      nq4=qlength4(1);
      pdebound(2,column)=m*ones(size(column));
      pdebound(3,column)=qlength1;
      pdebound(4,column)=qlength2;
      pdebound(5,column)=qlength3;
      pdebound(6,column)=qlength4;
      pdebound(7,column)=glength1;
      pdebound(8,column)=glength2;
      if m==2,
        nh1=hlength1(1); nr1=rlength1(1);
        nh2=hlength2(1); nr2=rlength2(1);
        nh3=hlength3(1);
        nh4=hlength4(1);
        pdebound(9,column)=hlength1;
        pdebound(10,column)=hlength2;
        pdebound(11,column)=hlength3;
        pdebound(12,column)=hlength4;
        pdebound(13,column)=rlength1;
        pdebound(14,column)=rlength2;
        pos=15;
        pdebound(pos:pos+nq1-1,column)=qstr1;
        pos=pos+nq1;
        pdebound(pos:pos+nq2-1,column)=qstr2;
        pos=pos+nq2;
        pdebound(pos:pos+nq3-1,column)=qstr3;
        pos=pos+nq3;
        pdebound(pos:pos+nq4-1,column)=qstr4;
        pos=pos+nq4;
        pdebound(pos:pos+ng1-1,column)=gstr1;
        pos=pos+ng1;
        pdebound(pos:pos+ng2-1,column)=gstr2;
        pos=pos+ng2;
        pdebound(pos:pos+nh1-1,column)=hstr1;
        pos=pos+nh1;
        pdebound(pos:pos+nh2-1,column)=hstr2;
        pos=pos+nh2;
        pdebound(pos:pos+nh3-1,column)=hstr3;
        pos=pos+nh3;
        pdebound(pos:pos+nh4-1,column)=hstr4;
        pos=pos+nh4;
        pdebound(pos:pos+nr1-1,column)=rstr1;
        pos=pos+nr1;
        pdebound(pos:pos+nr2-1,column)=rstr2;
```



```
   pos=12;
   pdebound(pos:pos+nq1-1,column)=qstr1;
   pos=pos+nq1;
   pdebound(pos:pos+nq2-1,column)=qstr2;
   pos=pos+nq2;
   pdebound(pos:pos+nq3-1,column)=qstr3;
   pos=pos+nq3;
   pdebound(pos:pos+nq4-1,column)=qstr4;
   pos=pos+nq4;
   pdebound(pos:pos+ng1-1,column)=gstr1;
   pos=pos+ng1;
   pdebound(pos:pos+ng2-1,column)=gstr2;
   pos=pos+ng2;
   pdebound(pos:pos+nh1-1,column)=hstr1;
   pos=pos+nh1;
   pdebound(pos:pos+nh2-1,column)=hstr2;
   pos=pos+nh2;
   pdebound(pos:pos+nr1-1,column)=rstr1;
  elseif m==0,
   pos=9;
   pdebound(pos:pos+nq1-1,column)=qstr1;
   pos=pos+nq1;
   pdebound(pos:pos+nq2-1,column)=qstr2;
   pos=pos+nq2;
   pdebound(pos:pos+nq3-1,column)=qstr3;
   pos=pos+nq3;
   pdebound(pos:pos+nq4-1,column)=qstr4;
   pos=pos+nq4;
   pdebound(pos:pos+ng1-1,column)=gstr1;
   pos=pos+ng1;
   pdebound(pos:pos+ng2-1,column)=gstr2;
  end
 end

 set(hbound,'UserData',pdebound)

 % alert PDETOOL that boundary conditions have changed
 flg_hndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEFileMenu');
 flags=get(flg_hndl,'UserData');
 flags(5)=1;                   % flag3=1
 flags(6)=0;                   % flag4=0
 set(flg_hndl,'UserData',flags)

 % deselect the boundary
 bndhndl=findobj(get(pde_fig,'Children'),'flat','Tag','PDEBoundMenu');
 set(findobj(get(bndhndl,'Children'),'flat','Tag','PDEBoundSpec'),...
   'UserData',[])

 delete(fig)
 drawnow
 pdeinfo
% case: cancel
elseif strcmp(action,'cancel')
 delete(gcf)
 drawnow
 pdeinfo
end
```

Interface Relaxation

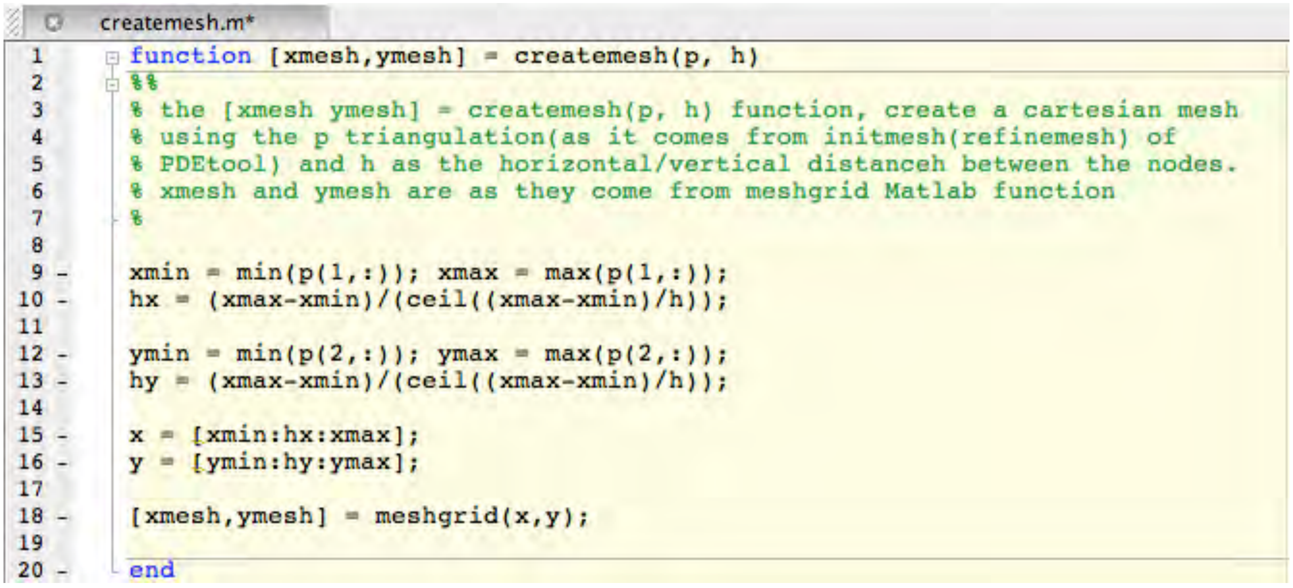
1. subdomaininfo.m



subdomaininfo.m*

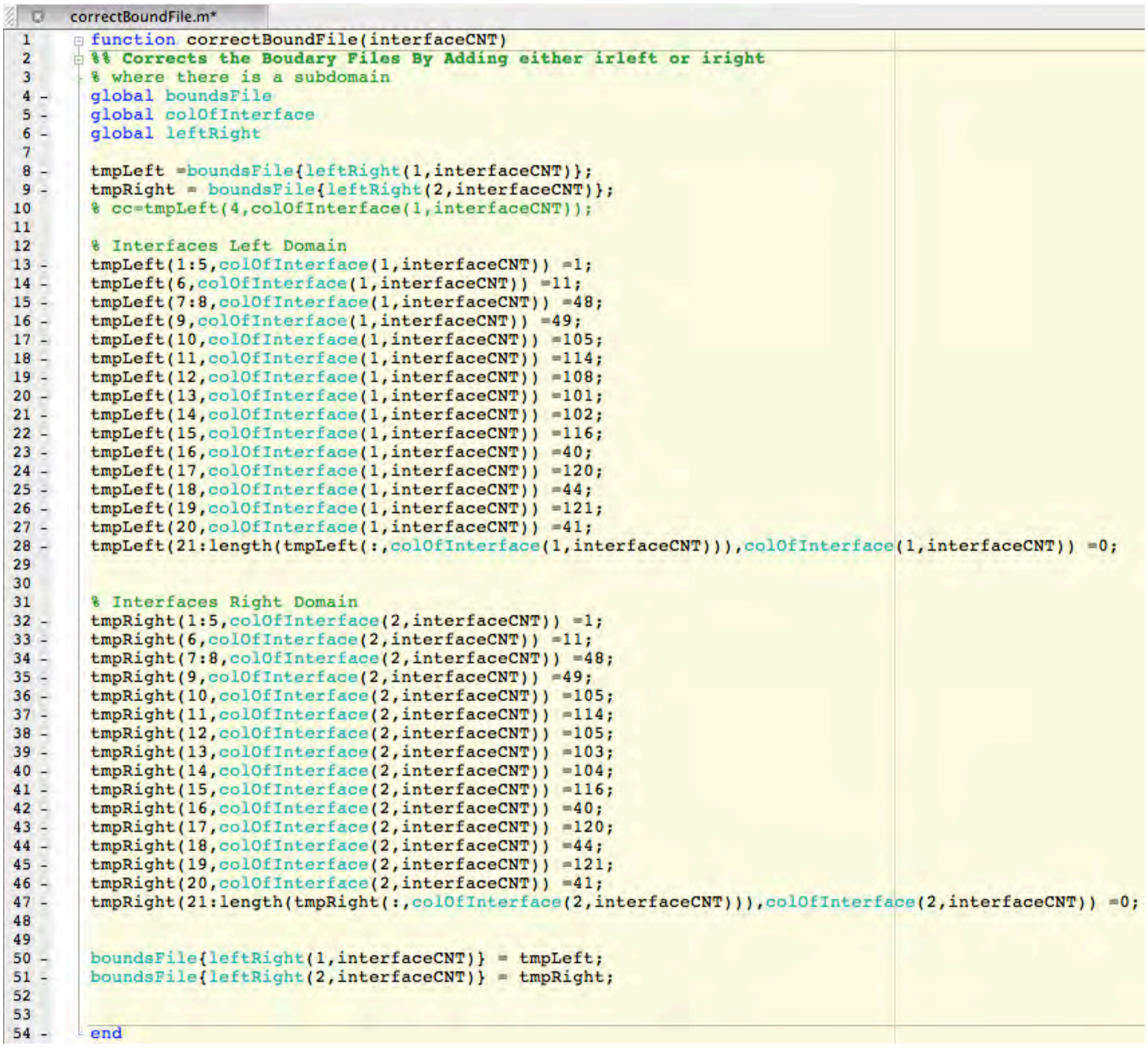
```
function [ numberOfLine,leftRight,irType,initialCondition,irParameters] = subdomaininfo(indexCurrent)
%% function that searches each domain for interfaces and then makes a
%%connection between interfaces and subdomains

    global splitGometryInCellArray;
    global irhndls;
    [geoR geoC]=size(splitGometryInCellArray);
    global boundsFile;
    irParameters =[];
    figs=findobj(allchild(0),'flat','Tag','PDETool');
% Find Global Geometry
    dl1 = splitGometryInCellArray{indexCurrent};
% Find number of interfaces
% Find in which colums of the geometry description matrix there is an
% interface
    bounds1 =find(dl1(7,:)~=0);
    bounds2 =find(dl1(6,:)~=0);

%this is the number of columns that contains the interface of the current
%domain
    intbounds = intersect(bounds1,bounds2);

%choose the column of geometry description table that contains the
%interface
interface = dl1(:,intbounds);

    [~,col] = size(interface);
    leftRight = ones(2,col);
     for i=1:col
         leftRight(1,i) = interface(6,i);
         leftRight(2,i) = interface(7,i);
     end

     for h=1:col
         tmp1 = splitGometryInCellArray{leftRight(1,h)};
         tmp2 = splitGometryInCellArray{leftRight(2,h)};
         [r1 c1] = size (tmp1);
         [r2 c2] = size (tmp2);

         for g=1:c1
             if interface(:,h) == tmp1(:,g)
                numberOfLine(1,h) = g;
                if indexCurrent ~= leftRight(1,h)
                    neighbours(h) = leftRight(1,h);
                    neighboursHndl(h) = irhndls(leftRight(1,h));
                end
             end
         end

         for g=1:c2
             if interface(:,h) == tmp2(:,g)
                numberOfLine(2,h) = g;
                if indexCurrent ~= leftRight(2,h)
                    neighbours(h) = leftRight(2,h);
                    neighboursHndl(h) = irhndls(leftRight(2,h));
                end
             end
         end
     end
%find IR Type, Initial condition and parameters for each interface
    tmp = boundsFile{indexCurrent};
    irType = tmp(2,intbounds(1));

    for i=1:col
        tmp2 = tmp(:,intbounds(i));
        sizeInitialCond= tmp(4,intbounds(i));
        sizeWparameter= tmp(5,intbounds(i));
        initialCondition{i} = transpose(char(tmp2(8:8+sizeInitialCond-1)));
        irParameters1 = mat2str(transpose(char(tmp2(8+sizeInitialCond:8+sizeInitialCond+sizeWparameter-1))));
        irParameters1 = irParameters1(1,2:length(irParameters1)-1);
        irParameters1 = str2double(irParameters1);
        irParameters = [irParameters  irParameters1];
    end


end %function end
```

2.initmesh.m © Panagiota Tsompanopoulou – All rights reserved


createmesh.m*

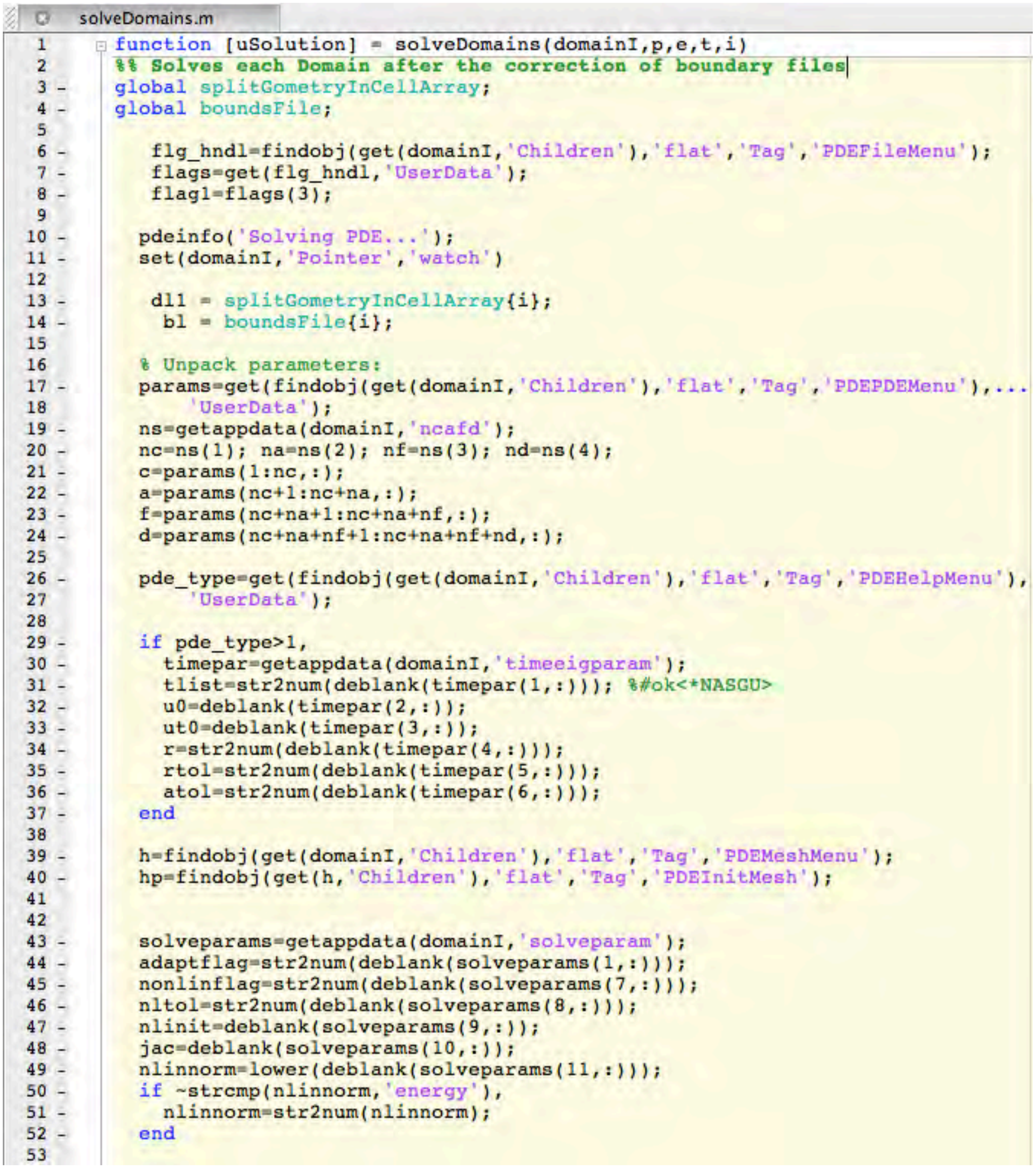
```matlab
function [xmesh,ymesh] = createmesh(p, h)
%%
% the [xmesh ymesh] = createmesh(p, h) function, create a cartesian mesh
% using the p triangulation(as it comes from initmesh(refinemesh) of
% PDEtool) and h as the horizontal/vertical distanceh between the nodes.
% xmesh and ymesh are as they come from meshgrid Matlab function
%

xmin = min(p(1,:)); xmax = max(p(1,:));
hx = (xmax-xmin)/(ceil((xmax-xmin)/h));

ymin = min(p(2,:)); ymax = max(p(2,:));
hy = (xmax-xmin)/(ceil((xmax-xmin)/h));

x = [xmin:hx:xmax];
y = [ymin:hy:ymax];

[xmesh,ymesh] = meshgrid(x,y);

end
```

3.correctBoundFile.m

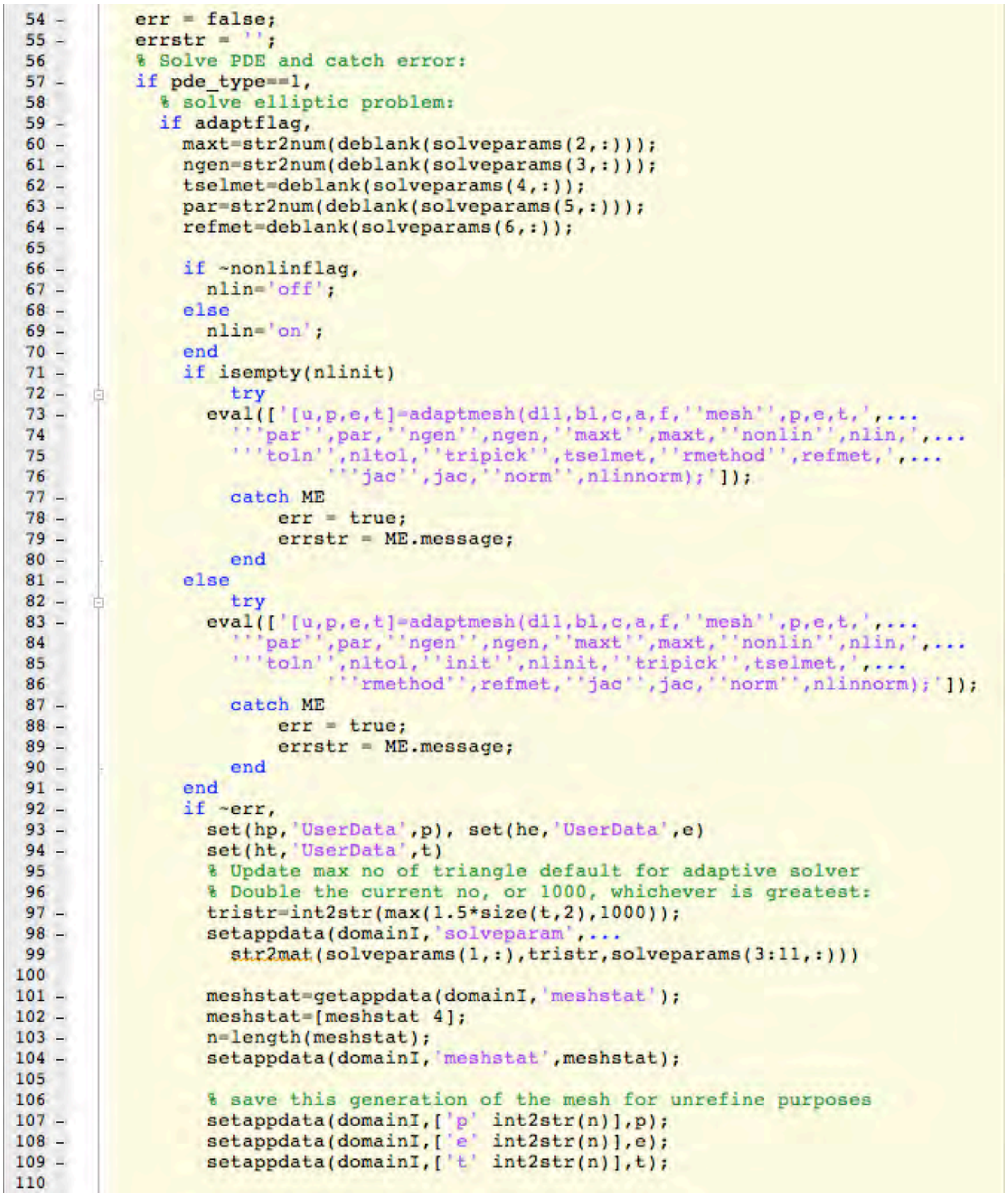

correctBoundFile.m*

```matlab
function correctBoundFile(interfaceCNT)
%% Corrects the Boudary Files By Adding either irleft or iright
% where there is a subdomain
global boundsFile
global colOfInterface
global leftRight

tmpLeft =boundsFile{leftRight(1,interfaceCNT)};
tmpRight = boundsFile{leftRight(2,interfaceCNT)};
% cc=tmpLeft(4,colOfInterface(1,interfaceCNT));

% Interfaces Left Domain
tmpLeft(1:5,colOfInterface(1,interfaceCNT)) =1;
tmpLeft(6,colOfInterface(1,interfaceCNT)) =11;
tmpLeft(7:8,colOfInterface(1,interfaceCNT)) =48;
tmpLeft(9,colOfInterface(1,interfaceCNT)) =49;
tmpLeft(10,colOfInterface(1,interfaceCNT)) =105;
tmpLeft(11,colOfInterface(1,interfaceCNT)) =114;
tmpLeft(12,colOfInterface(1,interfaceCNT)) =108;
tmpLeft(13,colOfInterface(1,interfaceCNT)) =101;
tmpLeft(14,colOfInterface(1,interfaceCNT)) =102;
tmpLeft(15,colOfInterface(1,interfaceCNT)) =116;
tmpLeft(16,colOfInterface(1,interfaceCNT)) =40;
tmpLeft(17,colOfInterface(1,interfaceCNT)) =120;
tmpLeft(18,colOfInterface(1,interfaceCNT)) =44;
tmpLeft(19,colOfInterface(1,interfaceCNT)) =121;
tmpLeft(20,colOfInterface(1,interfaceCNT)) =41;
tmpLeft(21:length(tmpLeft(:,colOfInterface(1,interfaceCNT))),colOfInterface(1,interfaceCNT)) =0;


% Interfaces Right Domain
tmpRight(1:5,colOfInterface(2,interfaceCNT)) =1;
tmpRight(6,colOfInterface(2,interfaceCNT)) =11;
tmpRight(7:8,colOfInterface(2,interfaceCNT)) =48;
tmpRight(9,colOfInterface(2,interfaceCNT)) =49;
tmpRight(10,colOfInterface(2,interfaceCNT)) =105;
tmpRight(11,colOfInterface(2,interfaceCNT)) =114;
tmpRight(12,colOfInterface(2,interfaceCNT)) =105;
tmpRight(13,colOfInterface(2,interfaceCNT)) =103;
tmpRight(14,colOfInterface(2,interfaceCNT)) =104;
tmpRight(15,colOfInterface(2,interfaceCNT)) =116;
tmpRight(16,colOfInterface(2,interfaceCNT)) =40;
tmpRight(17,colOfInterface(2,interfaceCNT)) =120;
tmpRight(18,colOfInterface(2,interfaceCNT)) =44;
tmpRight(19,colOfInterface(2,interfaceCNT)) =121;
tmpRight(20,colOfInterface(2,interfaceCNT)) =41;
tmpRight(21:length(tmpRight(:,colOfInterface(2,interfaceCNT))),colOfInterface(2,interfaceCNT)) =0;


boundsFile{leftRight(1,interfaceCNT)} = tmpLeft;
boundsFile{leftRight(2,interfaceCNT)} = tmpRight;


end
```

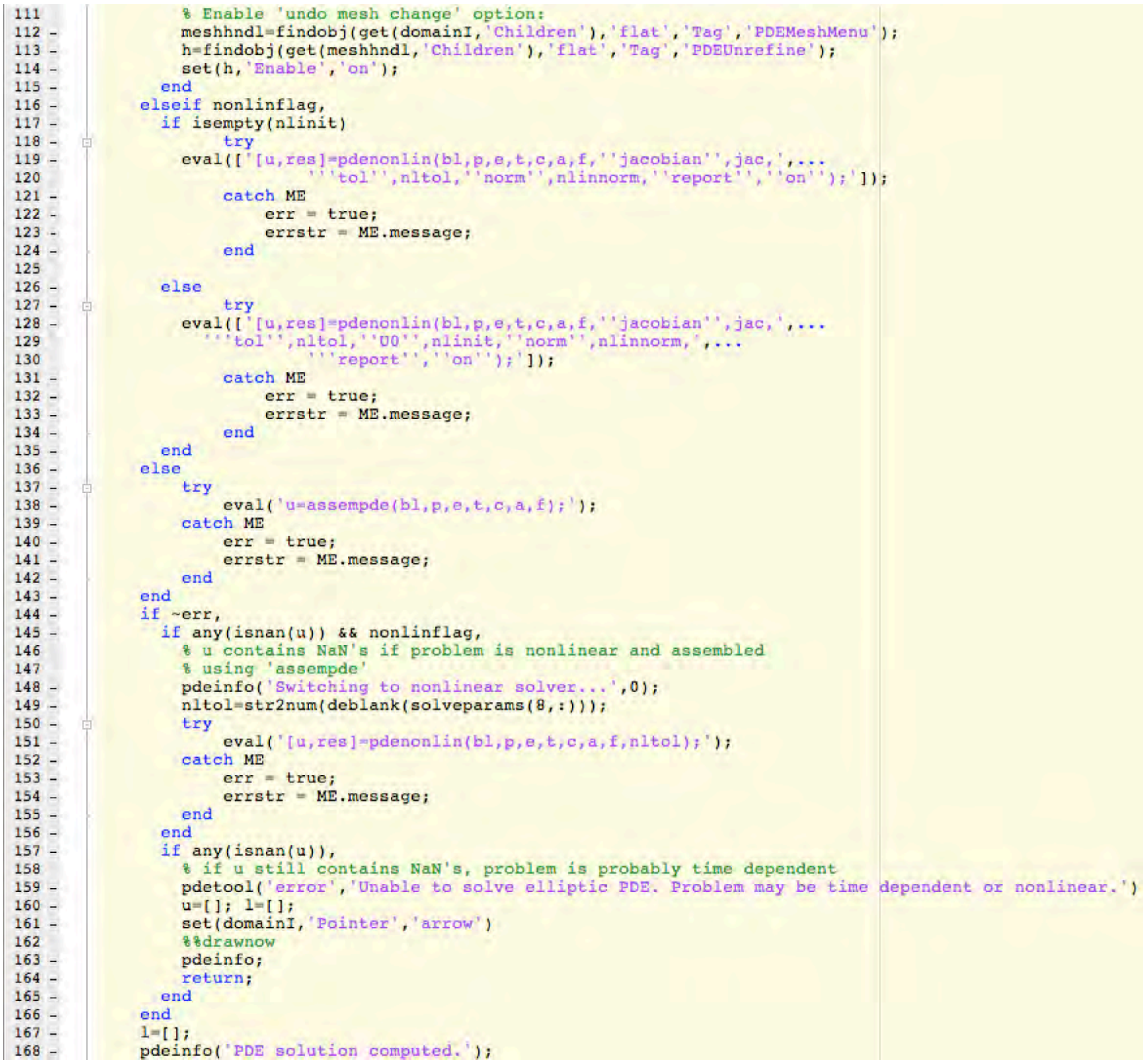
4.solveDomains.m


solveDomains.m

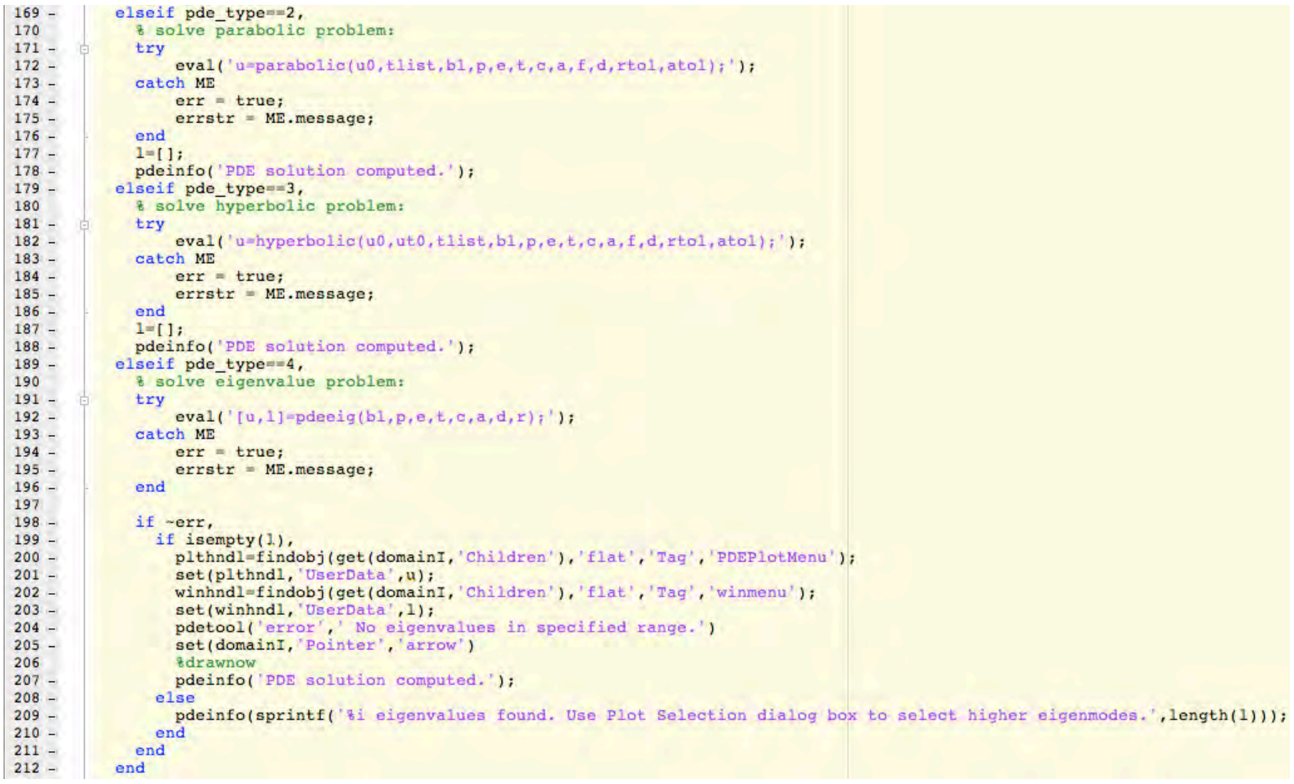
```
function [uSolution] = solveDomains(domainI,p,e,t,i)
%% Solves each Domain after the correction of boundary files
global splitGometryInCellArray;
global boundsFile;

   flg_hndl=findobj(get(domainI,'Children'),'flat','Tag','PDEFileMenu');
   flags=get(flg_hndl,'UserData');
   flag1=flags(3);

  pdeinfo('Solving PDE...');
  set(domainI,'Pointer','watch')

   dl1 = splitGometryInCellArray{i};
    bl = boundsFile{i};

  % Unpack parameters:
  params=get(findobj(get(domainI,'Children'),'flat','Tag','PDEPDEMenu'),...
      'UserData');
  ns=getappdata(domainI,'ncafd');
  nc=ns(1); na=ns(2); nf=ns(3); nd=ns(4);
  c=params(1:nc,:);
  a=params(nc+1:nc+na,:);
  f=params(nc+na+1:nc+na+nf,:);
  d=params(nc+na+nf+1:nc+na+nf+nd,:);

  pde_type=get(findobj(get(domainI,'Children'),'flat','Tag','PDEHelpMenu'),
      'UserData');

  if pde_type>1,
    timepar=getappdata(domainI,'timeeigparam');
    tlist=str2num(deblank(timepar(1,:))); %#ok<*NASGU>
    u0=deblank(timepar(2,:));
    ut0=deblank(timepar(3,:));
    r=str2num(deblank(timepar(4,:)));
    rtol=str2num(deblank(timepar(5,:)));
    atol=str2num(deblank(timepar(6,:)));
  end

  h=findobj(get(domainI,'Children'),'flat','Tag','PDEMeshMenu');
  hp=findobj(get(h,'Children'),'flat','Tag','PDEInitMesh');


  solveparams=getappdata(domainI,'solveparam');
  adaptflag=str2num(deblank(solveparams(1,:)));
  nonlinflag=str2num(deblank(solveparams(7,:)));
  nltol=str2num(deblank(solveparams(8,:)));
  nlinit=deblank(solveparams(9,:));
  jac=deblank(solveparams(10,:));
  nlinnorm=lower(deblank(solveparams(11,:)));
  if ~strcmp(nlinnorm,'energy'),
    nlinnorm=str2num(nlinnorm);
  end

```



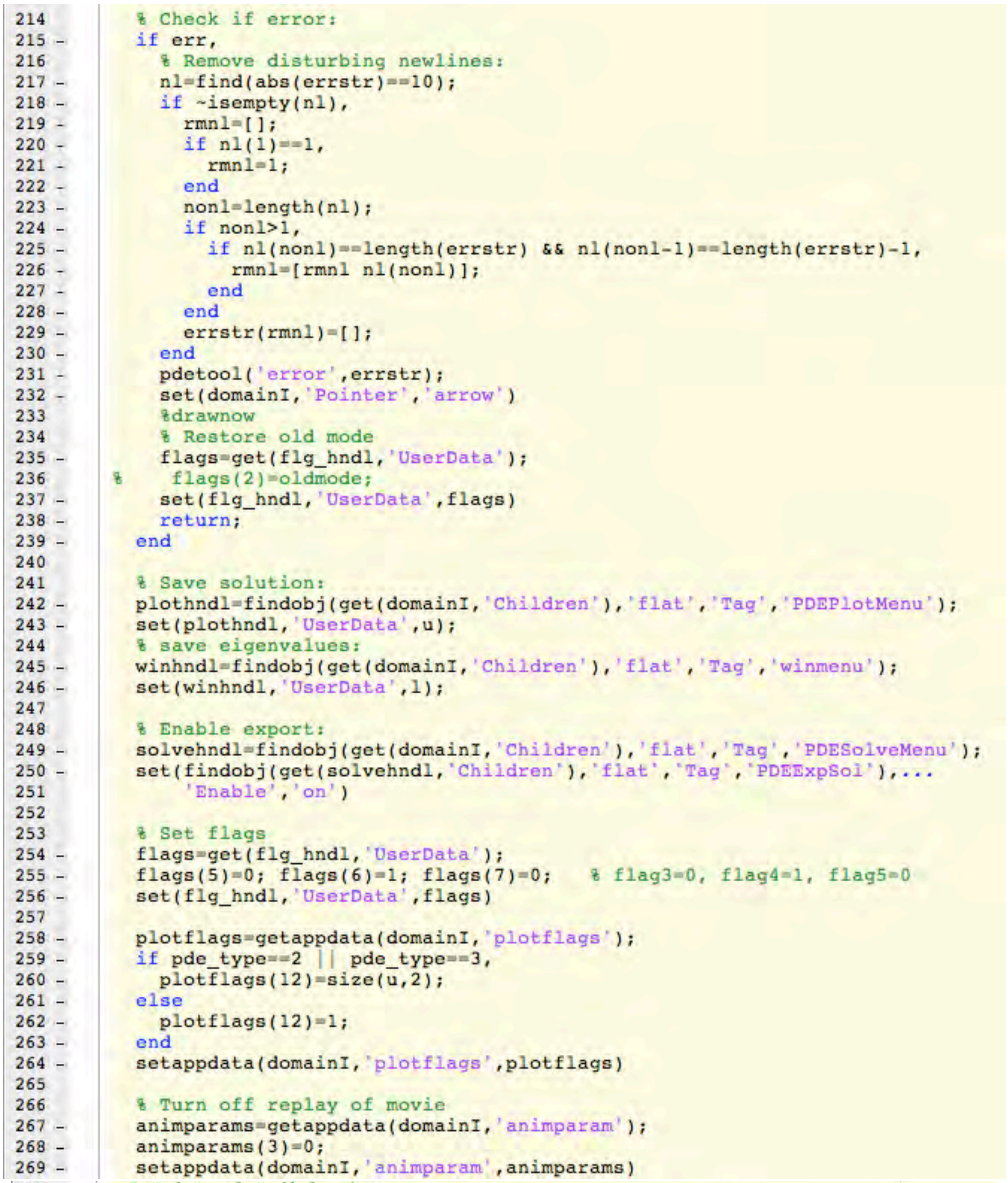
```
err = false;
errstr = '';
% Solve PDE and catch error:
if pde_type==1,
  % solve elliptic problem:
  if adaptflag,
    maxt=str2num(deblank(solveparams(2,:)));
    ngen=str2num(deblank(solveparams(3,:)));
    tselmet=deblank(solveparams(4,:));
    par=str2num(deblank(solveparams(5,:)));
    refmet=deblank(solveparams(6,:));

    if ~nonlinflag,
      nlin='off';
    else
      nlin='on';
    end
    if isempty(nlinit)
        try
      eval(['[u,p,e,t]=adaptmesh(dl1,bl,c,a,f,''mesh'',p,e,t,',...
        '''par'',par,''ngen'',ngen,''maxt'',maxt,''nonlin'',nlin,',...
        '''toln'',nltol,''tripick'',tselmet,''rmethod'',refmet,',...
                '''jac'',jac,''norm'',nlinnorm);']);
        catch ME
            err = true;
            errstr = ME.message;
        end
    else
        try
      eval(['[u,p,e,t]=adaptmesh(dl1,bl,c,a,f,''mesh'',p,e,t,',...
        '''par'',par,''ngen'',ngen,''maxt'',maxt,''nonlin'',nlin,',...
        '''toln'',nltol,''init'',nlinit,''tripick'',tselmet,',...
                '''rmethod'',refmet,''jac'',jac,''norm'',nlinnorm);']);
        catch ME
            err = true;
            errstr = ME.message;
        end
    end
    if ~err,
      set(hp,'UserData',p), set(he,'UserData',e)
      set(ht,'UserData',t)
      % Update max no of triangle default for adaptive solver
      % Double the current no, or 1000, whichever is greatest:
      tristr=int2str(max(1.5*size(t,2),1000));
      setappdata(domainI,'solveparam',...
        str2mat(solveparams(1,:),tristr,solveparams(3:11,:)))

      meshstat=getappdata(domainI,'meshstat');
      meshstat=[meshstat 4];
      n=length(meshstat);
      setappdata(domainI,'meshstat',meshstat);

      % save this generation of the mesh for unrefine purposes
      setappdata(domainI,['p' int2str(n)],p);
      setappdata(domainI,['e' int2str(n)],e);
      setappdata(domainI,['t' int2str(n)],t);
```



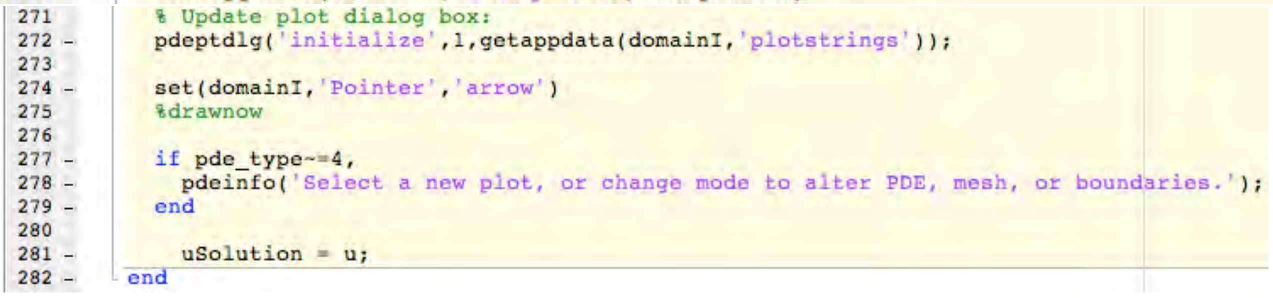
```
        % Enable 'undo mesh change' option:
        meshhndl=findobj(get(domainI,'Children'),'flat','Tag','PDEMeshMenu');
        h=findobj(get(meshhndl,'Children'),'flat','Tag','PDEUnrefine');
        set(h,'Enable','on');
      end
    elseif nonlinflag,
      if isempty(nlinit)
            try
        eval(['[u,res]=pdenonlin(bl,p,e,t,c,a,f,''jacobian'',jac,',...
                    '''tol'',nltol,''norm'',nlinnorm,''report'',''on'');']);
            catch ME
                err = true;
                errstr = ME.message;
            end

      else
            try
        eval(['[u,res]=pdenonlin(bl,p,e,t,c,a,f,''jacobian'',jac,',...
          '''tol'',nltol,''U0'',nlinit,''norm'',nlinnorm,',...
                    '''report'',''on'');']);
            catch ME
                err = true;
                errstr = ME.message;
            end
      end
    else
        try
            eval('u=assempde(bl,p,e,t,c,a,f);');
        catch ME
            err = true;
            errstr = ME.message;
        end
    end
    if ~err,
      if any(isnan(u)) && nonlinflag,
        % u contains NaN's if problem is nonlinear and assembled
        % using 'assempde'
        pdeinfo('Switching to nonlinear solver...',0);
        nltol=str2num(deblank(solveparams(8,:)));
        try
            eval('[u,res]=pdenonlin(bl,p,e,t,c,a,f,nltol);');
        catch ME
            err = true;
            errstr = ME.message;
        end
      end
      if any(isnan(u)),
        % if u still contains NaN's, problem is probably time dependent
        pdetool('error','Unable to solve elliptic PDE. Problem may be time dependent or nonlinear.')
        u=[]; l=[];
        set(domainI,'Pointer','arrow')
        %%drawnow
        pdeinfo;
        return;
      end
    end
    l=[];
    pdeinfo('PDE solution computed.');
```



```
elseif pde_type==2,
  % solve parabolic problem:
  try
      eval('u=parabolic(u0,tlist,bl,p,e,t,c,a,f,d,rtol,atol);');
  catch ME
      err = true;
      errstr = ME.message;
  end
  l=[];
  pdeinfo('PDE solution computed.');
elseif pde_type==3,
  % solve hyperbolic problem:
  try
      eval('u=hyperbolic(u0,ut0,tlist,bl,p,e,t,c,a,f,d,rtol,atol);');
  catch ME
      err = true;
      errstr = ME.message;
  end
  l=[];
  pdeinfo('PDE solution computed.');
elseif pde_type==4,
  % solve eigenvalue problem:
  try
      eval('[u,l]=pdeeig(bl,p,e,t,c,a,d,r);');
  catch ME
      err = true;
      errstr = ME.message;
  end

  if ~err,
    if isempty(l),
      plthndl=findobj(get(domainI,'Children'),'flat','Tag','PDEPlotMenu');
      set(plthndl,'UserData',u);
      winhndl=findobj(get(domainI,'Children'),'flat','Tag','winmenu');
      set(winhndl,'UserData',l);
      pdetool('error',' No eigenvalues in specified range.')
      set(domainI,'Pointer','arrow')
      %drawnow
      pdeinfo('PDE solution computed.');
    else
      pdeinfo(sprintf('%i eigenvalues found. Use Plot Selection dialog box to select higher eigenmodes.',length(l)));
    end
  end
end
```

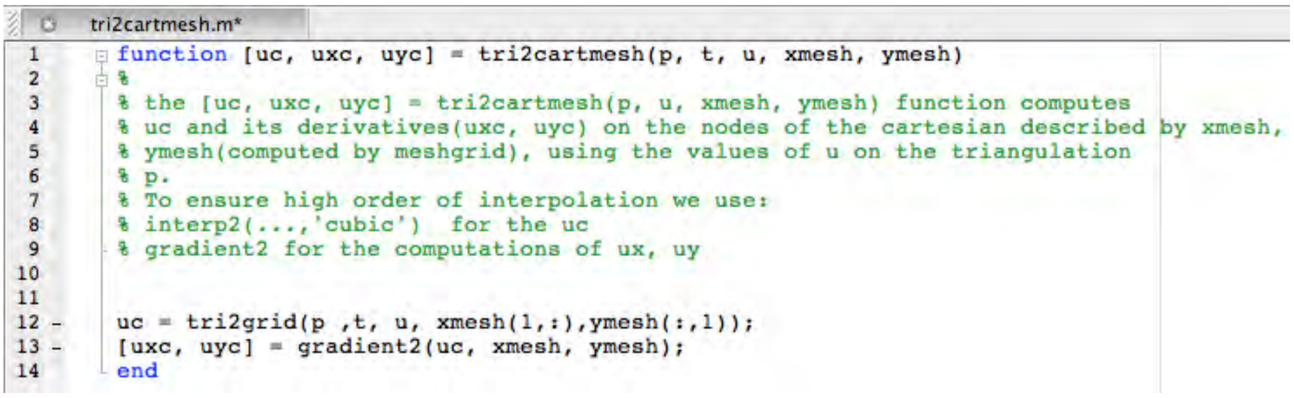


```
  % Check if error:
  if err,
    % Remove disturbing newlines:
    nl=find(abs(errstr)==10);
    if ~isempty(nl),
      rmnl=[];
      if nl(1)==1,
        rmnl=1;
      end
      nonl=length(nl);
      if nonl>1,
        if nl(nonl)==length(errstr) && nl(nonl-1)==length(errstr)-1,
          rmnl=[rmnl nl(nonl)];
        end
      end
      errstr(rmnl)=[];
    end
    pdetool('error',errstr);
    set(domainI,'Pointer','arrow')
    %drawnow
    % Restore old mode
    flags=get(flg_hndl,'UserData');
%    flags(2)=oldmode;
    set(flg_hndl,'UserData',flags)
    return;
  end

  % Save solution:
  plothndl=findobj(get(domainI,'Children'),'flat','Tag','PDEPlotMenu');
  set(plothndl,'UserData',u);
  % save eigenvalues:
  winhndl=findobj(get(domainI,'Children'),'flat','Tag','winmenu');
  set(winhndl,'UserData',1);

  % Enable export:
  solvehndl=findobj(get(domainI,'Children'),'flat','Tag','PDESolveMenu');
  set(findobj(get(solvehndl,'Children'),'flat','Tag','PDEExpSol'),...
      'Enable','on')

  % Set flags
  flags=get(flg_hndl,'UserData');
  flags(5)=0; flags(6)=1; flags(7)=0;   % flag3=0, flag4=1, flag5=0
  set(flg_hndl,'UserData',flags)

  plotflags=getappdata(domainI,'plotflags');
  if pde_type==2 || pde_type==3,
    plotflags(12)=size(u,2);
  else
    plotflags(12)=1;
  end
  setappdata(domainI,'plotflags',plotflags)

  % Turn off replay of movie
  animparams=getappdata(domainI,'animparam');
  animparams(3)=0;
  setappdata(domainI,'animparam',animparams)
```



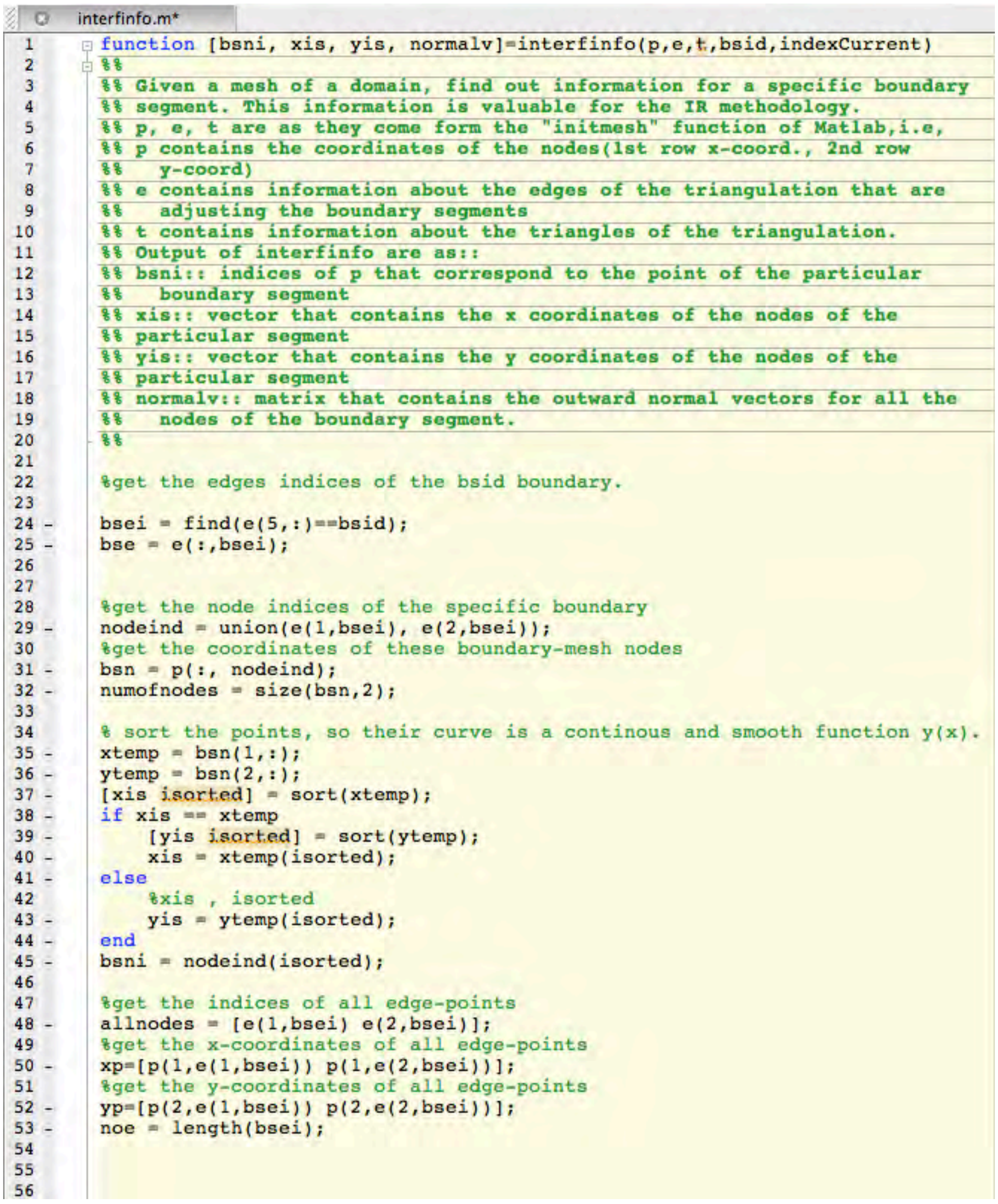
```
  % Update plot dialog box:
  pdeptdlg('initialize',1,getappdata(domainI,'plotstrings'));

  set(domainI,'Pointer','arrow')
  %drawnow

  if pde_type~=4,
    pdeinfo('Select a new plot, or change mode to alter PDE, mesh, or boundaries.');
  end

    uSolution = u;
end
```

5. ***tri2cartmesh.m*** © Panagiota Tsompanopoulou – All rights reserved

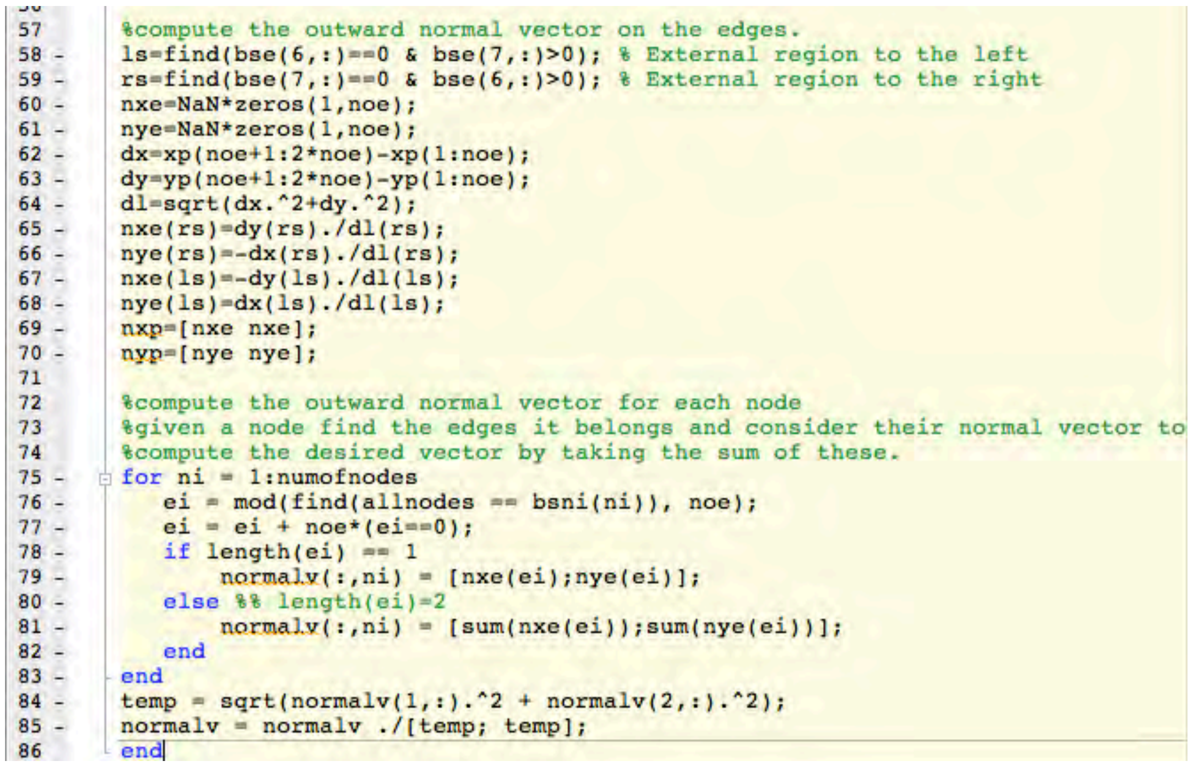


```
function [uc, uxc, uyc] = tri2cartmesh(p, t, u, xmesh, ymesh)
%
% the [uc, uxc, uyc] = tri2cartmesh(p, u, xmesh, ymesh) function computes
% uc and its derivatives(uxc, uyc) on the nodes of the cartesian described by xmesh,
% ymesh(computed by meshgrid), using the values of u on the triangulation
% p.
% To ensure high order of interpolation we use:
% interp2(...,'cubic')  for the uc
% gradient2 for the computations of ux, uy


uc = tri2grid(p ,t, u, xmesh(1,:),ymesh(:,1));
[uxc, uyc] = gradient2(uc, xmesh, ymesh);
end
```

6.interfinfo.m © Panagiota Tsompanopoulou – All rights reserved

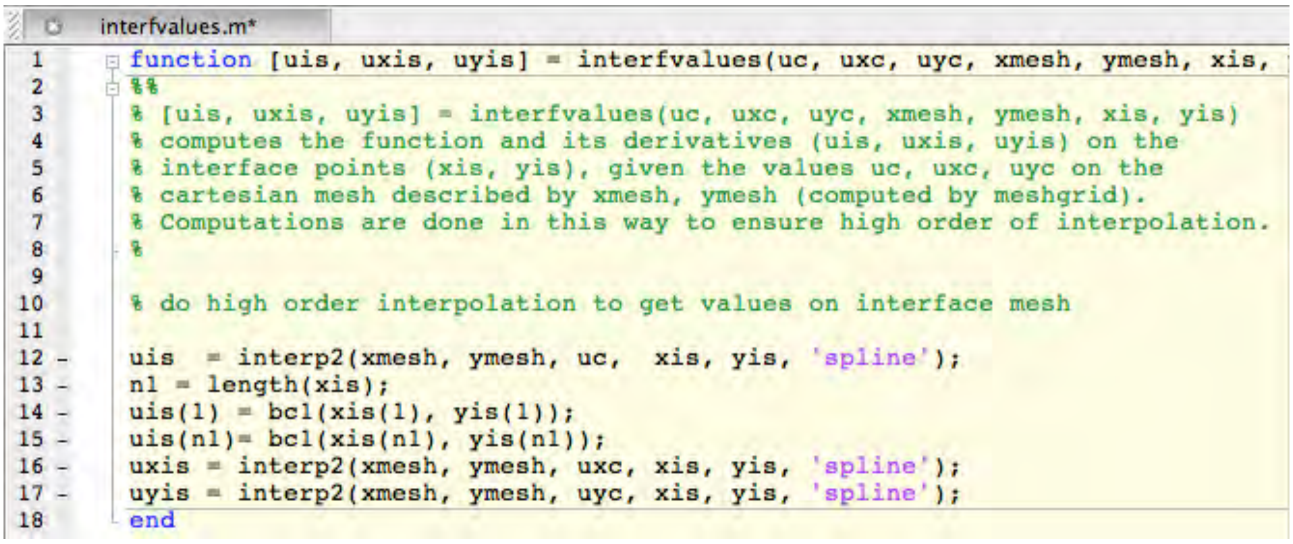


```
function [bsni, xis, yis, normalv]=interfinfo(p,e,t,bsid,indexCurrent)
%%
%% Given a mesh of a domain, find out information for a specific boundary
%% segment. This information is valuable for the IR methodology.
%% p, e, t are as they come form the "initmesh" function of Matlab,i.e,
%% p contains the coordinates of the nodes(1st row x-coord., 2nd row
%%    y-coord)
%% e contains information about the edges of the triangulation that are
%%    adjusting the boundary segments
%% t contains information about the triangles of the triangulation.
%% Output of interfinfo are as::
%% bsni:: indices of p that correspond to the point of the particular
%%    boundary segment
%% xis:: vector that contains the x coordinates of the nodes of the
%% particular segment
%% yis:: vector that contains the y coordinates of the nodes of the
%% particular segment
%% normalv:: matrix that contains the outward normal vectors for all the
%%    nodes of the boundary segment.
%%

%get the edges indices of the bsid boundary.

bsei = find(e(5,:)==bsid);
bse = e(:,bsei);


%get the node indices of the specific boundary
nodeind = union(e(1,bsei), e(2,bsei));
%get the coordinates of these boundary-mesh nodes
bsn = p(:, nodeind);
numofnodes = size(bsn,2);

% sort the points, so their curve is a continous and smooth function y(x).
xtemp = bsn(1,:);
ytemp = bsn(2,:);
[xis isorted] = sort(xtemp);
if xis == xtemp
    [yis isorted] = sort(ytemp);
    xis = xtemp(isorted);
else
    %xis , isorted
    yis = ytemp(isorted);
end
bsni = nodeind(isorted);

%get the indices of all edge-points
allnodes = [e(1,bsei) e(2,bsei)];
%get the x-coordinates of all edge-points
xp=[p(1,e(1,bsei)) p(1,e(2,bsei))];
%get the y-coordinates of all edge-points
yp=[p(2,e(1,bsei)) p(2,e(2,bsei))];
noe = length(bsei);
```



```
%compute the outward normal vector on the edges.
ls=find(bse(6,:)==0 & bse(7,:)>0); % External region to the left
rs=find(bse(7,:)==0 & bse(6,:)>0); % External region to the right
nxe=NaN*zeros(1,noe);
nye=NaN*zeros(1,noe);
dx=xp(noe+1:2*noe)-xp(1:noe);
dy=yp(noe+1:2*noe)-yp(1:noe);
dl=sqrt(dx.^2+dy.^2);
nxe(rs)=dy(rs)./dl(rs);
nye(rs)=-dx(rs)./dl(rs);
nxe(ls)=-dy(ls)./dl(ls);
nye(ls)=dx(ls)./dl(ls);
nxp=[nxe nxe];
nyp=[nye nye];

%compute the outward normal vector for each node
%given a node find the edges it belongs and consider their normal vector to
%compute the desired vector by taking the sum of these.
for ni = 1:numofnodes
    ei = mod(find(allnodes == bsni(ni)), noe);
    ei = ei + noe*(ei==0);
    if length(ei) == 1
        normalv(:,ni) = [nxe(ei);nye(ei)];
    else %% length(ei)=2
        normalv(:,ni) = [sum(nxe(ei));sum(nye(ei))];
    end
end
temp = sqrt(normalv(1,:).^2 + normalv(2,:).^2);
normalv = normalv ./[temp; temp];
end
```

7.interfvalues.m - © Panagiota Tsompanopoulou – All rights reserved



interfvalues.m*

```
function [uis, uxis, uyis] = interfvalues(uc, uxc, uyc, xmesh, ymesh, xis,
%%
% [uis, uxis, uyis] = interfvalues(uc, uxc, uyc, xmesh, ymesh, xis, yis)
% computes the function and its derivatives (uis, uxis, uyis) on the
% interface points (xis, yis), given the values uc, uxc, uyc on the
% cartesian mesh described by xmesh, ymesh (computed by meshgrid).
% Computations are done in this way to ensure high order of interpolation.
%

% do high order interpolation to get values on interface mesh

uis  = interp2(xmesh, ymesh, uc,  xis, yis, 'spline');
nl = length(xis);
uis(1) = bcl(xis(1), yis(1));
uis(nl)= bcl(xis(nl), yis(nl));
uxis = interp2(xmesh, ymesh, uxc, xis, yis, 'spline');
uyis = interp2(xmesh, ymesh, uyc, xis, yis, 'spline');
end
```