Lecture Notes for the Course on
NUMERICAL METHODS FOR NUCLEAR REACTORS

Prof. WALTER AMBROSINI

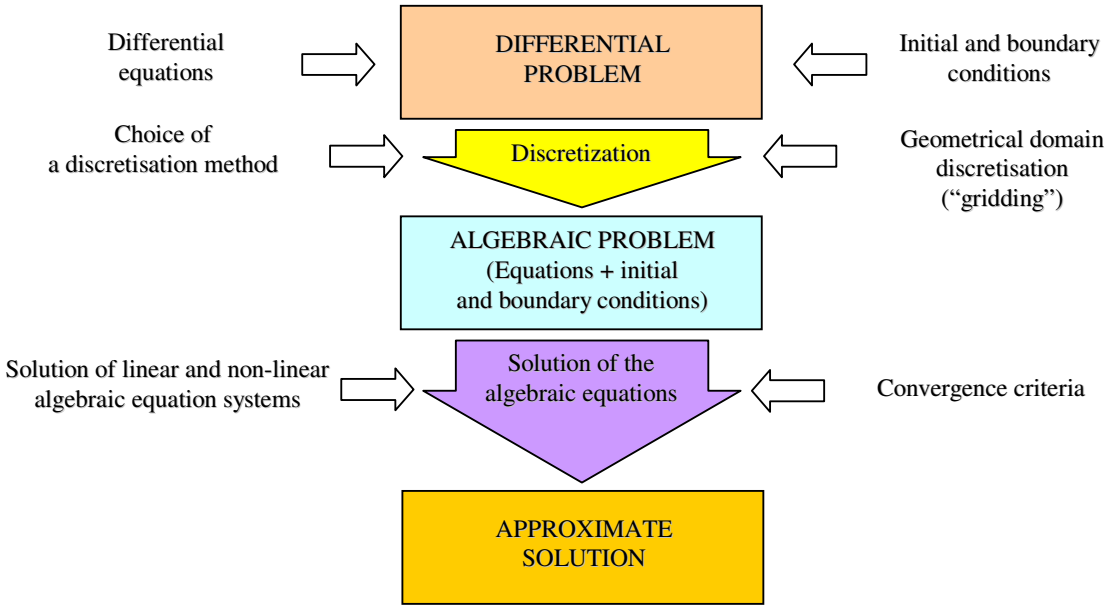
University of Pisa, Italy

Unit-3 – More on Discretisation Methods and Sample Applications to Eigenvalue Problems

NOTICE: These notes were prepared by Prof. Ambrosini mainly **on the basis of the material adopted by Prof. Bruno Montagnini for the lectures he held up to years ago**, when he left to Prof. Ambrosini the charge of the course held for the Degree in Nuclear Engineering at the University of Pisa. This material is freely distributed to Course attendees or to anyone else requesting it. It has not the worth of a textbook and it is not intended to be an official publication. It was conceived as the notes that the teacher himself would take of his own lectures in the paradoxical case he could be both teacher and student at the same time (sometimes space and time stretch and fold in strange ways). It is also used as slides to be projected during lectures to assure a minimum of uniform, constant quality lecturing, regardless of the teacher's good and bad days. As such, the material contains reference to classical textbooks and material whose direct reading is warmly recommended to students for a more accurate understanding. In the attempt to make these notes as original as feasible and reasonable, considering their purely educational purpose, most of the material has been completely re-interpreted in the teacher's own view and personal preferences about notation. In this effort, errors in details may have been introduced which will be promptly corrected in further versions after discovery. Requests of clarification, suggestions, complaints or even sharp judgements in relation to this material can be directly addressed to Prof. Ambrosini at the e-mail address: walter.ambrosini@ing.unipi.it

DISCRETISATION METHODS

- **In the previous units we assumed a rather heuristic way to discretise equations by "finite difference" or "finite volume" approaches**
- **Let us now come back on this aspect, by looking more carefully to the different possible choices available to obtain the approximate solution of a partial differential equation problem**
- **The figure below summarises the main steps to be performed in the process of solving partial differential equation problems by suitable discretization schemes**
- **The characteristics of the main available discretization methods will be summarised in the following**



- **Generally speaking we have three main techniques of discretizing neutron diffusion and transport equations:**
  - **the *finite difference method*, i.e., the substitution of partial derivatives with difference expressions (already considered);**
  - **the *finite volume method*, i.e., writing equations in integral form over control volumes (already considered and to be further applied later);**
  - **the *finite element methods*, i.e., making use of the "weighted residuals method" to get more accurate local approximations, as in the "coarse-mesh method" that we will describe below.**

WEIGHTED RESIDUALS METHOD

We search for the solution of the diffusion (or any other) partial differential equation written in the form

$$L\phi + S = 0$$

that is a compact way to write

$$div\, D\, grad\, \phi - \Sigma_a \phi + S = 0$$

with some assigned boundary conditions.

We now search for an *approximate solution* of this equation in a narrower class of functions than generally eligible for the application of the differential operator:

$$\phi_{app}\left(\vec{r}\right) = \sum_{j=1}^{N} c_j u_j\left(\vec{r}\right)$$

where:

- $u_i(\vec{r})$ **are trial functions that are sufficiently regular and must satisfy the boundary conditions;**
- c_j **= appropriate coefficients-**

The "residual" of the differential equation, obtained upon substitution in it of the approximate solution, will be generally different from zero

$$L\phi_{app} + S = R\left(\vec{r}\right) \neq 0$$

In order to obtain a good approximation of the solution of the differential equation, we can impose that the residual be sufficiently small in some integral sense.

In this purpose, we search for *N* "weighting functions" $w_i(\vec{r})$ **imposing that the weighted residual be zero in the average over the considered volume:**

$$\int_V R(\vec{r})\, w_i(\vec{r})\, dV = 0 \quad (i = 1, \ldots, N)$$

meaning that

$$\int_V \left(L\phi_{app} + S \right) w_i(\vec{r})\, dV = 0 \qquad (i = 1, \ldots, N)$$

Making use of a notation of "inner product" between functions (as in vector spaces), we have

$$(R, w_i) = \int_V R(\vec{r})\, w_i(\vec{r})\, dV = \left(L \sum_{j=1}^{N} c_j u_j, w_i \right) + (S, w_i) = 0$$

Since the trial functions and the weighting functions are known, imposing this relationship implies to select an appropriate value of the coefficients of the expansion in the approximation.

In fact, a linear system in the unknown coefficients is thus obtained:

$$\sum_{j=1}^{N} c_j \left(L u_j, w_i \right) + (S, w_i) = 0$$

$$\Rightarrow \qquad \sum_{j=1}^{N} c_j a_{ij} + s_i = 0 \qquad (i = 1, \ldots, N)$$

Generalising the concept of orthogonality between vectors to the case of functions, we can say that the residual is imposed to be "orthogonal" to each weighting function. So, increasing the number of weighing functions, we can obtain more and more accurate approximations (i.e., "orthogonal to many functions").

The choice of "trial functions" is generally left to good practice:

- **they should be simple enough to be differentiated and integrated at ease;**
- **they should provide trends similar to the expected exact solution at least locally.**

For the above reasons, *low order polynomials* are often chosen in this purpose, but their selection is not mandatory.

On the other hand, the selection of *weighting functions* characterizes the method according to classical choices

- **in the *GALERKIN METHOD*, the "weighting functions" are the same as the "trial functions"**

$$w_i\left(\vec{r}\right) = u_i\left(\vec{r}\right)$$

in the case of a polynomial of n-th degree, for instance, it is:

$$\phi_{app}\left(x\right) = \sum_{j=0}^{N} c_j u_j\left(x\right) = c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n$$

and a Galerking weighting can make use of

$$w_i\left(x\right) = x^k \qquad \left(k = 0, \ldots, N\right)$$

- **in the *SUBDOMAIN METHOD,* it is assumed that the weighting function is "unity" in the selected subdomain:**

$$w_i\left(\vec{r}\right) = 1 \qquad \vec{r} \in V_m \qquad \textbf{and} \quad w_i\left(\vec{r}\right) = 0 \qquad \vec{r} \notin V_m$$

In the case of neutron diffusion this is equivalent to impose that the neutronic balance is satisfied in a global (integral) sense in the volume

In fact:

$$\int_{V_m} \left(div\, D\, grad\, \phi_{app} - \Sigma_a \phi_{app} + S \right) dV = 0$$

$$\underbrace{-\int_{S_m} -D\, grad\, \phi_{app} \cdot \vec{n}\; dS}_{leakage} \underbrace{-\int_{V_m} \Sigma_a \phi_{app}\; dV}_{absorption} + \underbrace{\int_{V_m} S\; dV}_{source} = 0$$

This is an important requirement that should be required to any numerical scheme:

the integral balance of neutrons should be satisfied within each volume by the adopted approximation

It is moreover necessary to impose that
neutron balance is satisfied everywhere in the whole domain by imposing appropriate continuity conditions of current at the interfaces.

- **in the *COLLOCATION METHOD,* a Dirac's "delta function" is used for weighting**

$$w_i\left(\vec{r}\right) = \delta\left(\vec{r} - \vec{r}_i\right)$$

This is equivalent to impose that the residual is zero at a certain point $\vec{r}_i$

$$\int_V \left(L\phi_{app} + S \right) \delta\left(\vec{r} - \vec{r}_i\right)\, dV = R\left(\vec{r}_i\right) = 0$$

This means that *the partial differential equation is solved exactly at that location by the approximating function*: of course, this <u>does not</u> mean that the solution is exact in that place

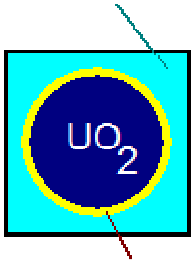
SAMPLE APPLICATION OF THE WEIGHTED RESIDUAL METHOD: "COARSE MESH" METHODS

Reactor calculations with large meshes are required in static and dynamic reactor analyses

The considerable complication of composition of nuclear reactors requires to perform calculations after convenient homogenization of properties at different levels

1. <u>Cell level homogenisation</u>

Coolant moderator

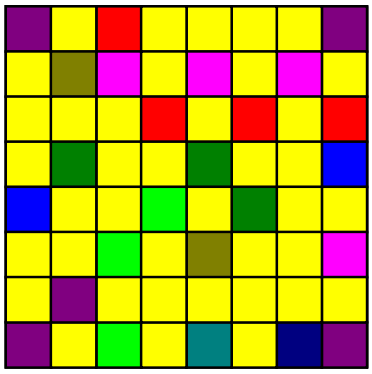


Cladding

- **The rod is considered together with the moderator/coolant assigned to it and to the cladding**
- **Neutron transport codes are used to obtain the nuclear parameters with few energy groups**
- **Many calculations are needed to consider the multiple values of enrichment, burn-up, burnable poisons, etc..**

2. Fuel assembly level homogenisation

- **The fuel element is then considered as obtained by different homogenised cells**



- **Actually an assembly calculation can be made even starting from the original more complex cell structure**

Making use of transport codes, the few group constants are calculated for the different axial locations where the composition is known

The *few group* parameters for each large "node" are so obtained as a function of:

- **composition of the element box;**
- **moderator temperature;**
- **void fraction;**
- **burn-up;**
- **fuel temperature.**

These "libraries" of tables are used to obtain by interpolation the parameters applicable to each case, in both static and dynamic calculations

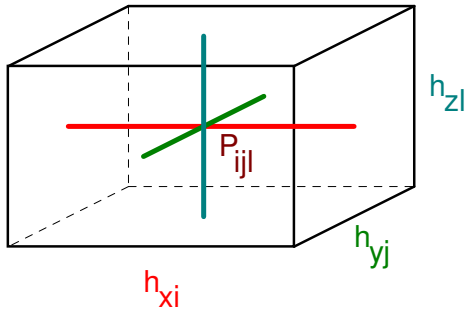
The *"coarse-mesh"* methods are used to perform calculations in 3D on the whole reactor on the basis of the obtained parameters

They are more efficient than the finite difference methods since they allow the use of a lower number of nodes with the same accuracy

This technique is described here since it was developed in the past at the University of Pisa. Alternative techniques are used in specific codes set up in different research groups

QUABOX Method

- **The reactor is subdivided into parallelepiped volumes centred on the points in which it is chosen to calculate the neutron flux**



- **Each node is considered as *"homogeneous"* in terms of material and the neutron flux is approximated in it by a second order polynomial in the different directions**

$$\phi(x,y,z) = \phi(\xi,\eta,\zeta) = \phi_{ijl}\,(1 + a_{x,ijl}\,\xi + b_{x,ijl}\,\xi^2 + a_{y,ijl}\,\eta + b_{y,ijl}\,\eta^2 + a_{z,ijl}\,\zeta + b_{z,ijl}\,\zeta^2)$$

where the *local coordinates* are defined as

$$\xi = \frac{x - x_i}{h_{xi}} \qquad \eta = \frac{y - y_j}{h_{yj}} \qquad \zeta = \frac{z - z_l}{h_{zl}}$$

[-1/2,1/2] and ϕ_{ijl} is the neutron flux in the node centre.

The "a" and "b" coefficients are related to the values of the neutron flux at the interfaces:

$$\phi_{i-\frac{1}{2}.j.l} = \phi_{ijl}\left(1 - \frac{a_x}{2} + \frac{b_x}{4}\right)$$

$$\phi_{i+\frac{1}{2}.j.l} = \phi_{ijl}\left(1 + \frac{a_x}{2} + \frac{b_x}{4}\right)$$

This allows calculating their values *in terms of interface fluxes*

$$a_x = \frac{\phi_{i+\frac{1}{2}.j.l} - \phi_{i-\frac{1}{2}.j.l}}{\phi_{ijl}} \qquad b_x = 2\,\frac{\phi_{i+\frac{1}{2}.j.l} - 2\,\phi_{ijl} + \phi_{i-\frac{1}{2}.j.l}}{\phi_{ijl}}$$

with similar relations holding for the y and z directions.

It is then imposed that the neutron balance is satisfied in the node: this corresponds to the *subdomain method*:

$$\int_{S_{ijl}} D\ \mathrm{grad}\,\phi \cdot \vec{n}\ dS + \int_{V_{ijl}} \left(\frac{\nu\,\Sigma_f}{k} - \Sigma_a\right)\phi\ dV = 0$$

These integrals are evaluated on the basis of the quadratic polynomial, obtaining:

$$2\,D_{ijl}\left(\frac{b_x}{2h_{xi}^2} + \frac{b_y}{2h_{yj}^2} + \frac{b_z}{2h_{zl}^2}\right)\phi_{ijl} +$$

$$+\left(\frac{\nu\Sigma_{f.ijl}}{k} - \Sigma_{a.ijl}\right)\left(1 + \frac{b_x + b_y + b_z}{12}\right)\phi_{ijl} = 0$$

Remembering the above defined expression for the coefficients of the polynomial, *a seven point formula* is obtained, relating the interfacial fluxes to the centre flux.

$$\left[D_{ijl} + \frac{h_{xi}^2}{24}\left(\frac{\nu\Sigma_{f.ijl}}{k} - \Sigma_{a.ijl}\right)\right]\frac{\phi_{i+½.j.l} - 2\,\phi_{ijl} + \phi_{i-½.j.l}}{(h_{xi}/2)^2}$$

$$+\left[D_{ijl} + \frac{h_{yj}^2}{24}\left(\frac{\nu\Sigma_{f.ijl}}{k} - \Sigma_{a.ijl}\right)\right]\frac{\phi_{i.j+½.l} - 2\,\phi_{ijl} + \phi_{i.j-½.l}}{(h_{yj}/2)^2}$$

$$+\left[D_{ijl} + \frac{h_{zl}^2}{24}\left(\frac{\nu\Sigma_{f.ijl}}{k} - \Sigma_{a.ijl}\right)\right]\frac{\phi_{i.j.l+½} - 2\,\phi_{ijl} + \phi_{i.j.l-½}}{(h_{zl}/2)^2} \qquad (*)$$

$$+\left(\frac{\nu\Sigma_{f.ijl}}{k} - \Sigma_{a.ijl}\right)\phi_{ijl} = 0$$

It must be now noted that this expression represents the neutron balance *"within the node"*. In order to satisfy the overall balance everywhere in the calculation domain (i.e., in the system containing all the parallelepiped volumes), it must be completed by conditions expressing the "neutron balance at the interfaces between the nodes":

we have six current continuity equations as the following one

$$- D_{i.j.l}\left[\frac{\partial\phi}{\partial x}\right]_{x_{i+½}}^{-} = - D_{i+1.j.l}\left[\frac{\partial\phi}{\partial x}\right]_{x_{i+1-½}}^{+}$$

$$\frac{D_{i.j.l}}{h_{xi}}\left(3\,\phi_{i+½.j.l} - 4\,\phi_{i.j.l} + \phi_{i-½.j.l}\right) =$$

$$= \frac{D_{i+1.j.l}}{h_{xi+1}}\left(-3\,\phi_{i+1-½.j.l} + 4\,\phi_{i+1.j.l} - 3\,\phi_{i+1+½.j.l}\right)$$

The other 5 are quite similar.

These equations serve *to assure the overall neutron balance in the domain* and allow "to express the interface

fluxes in terms of node centred fluxes", being the true unknowns of the problem:

$$\phi_{i+½,j,l} = \frac{4\left(1 - \frac{1}{4}\frac{\phi_{i-½,j,l}}{\phi_{i,j,l}}\right)}{3\left(1 + \frac{h_{xi}\, D_{i+1,j,l}}{h_{xi+1}\, D_{i,j,l}}\right)}\phi_{i,j,l} + \frac{4\left(1 - \frac{1}{4}\frac{\phi_{i+1+½,j,l}}{\phi_{i+1,j,l}}\right)}{3\left(1 + \frac{h_{xi+1}\, D_{i,j,l}}{h_{xi}\, D_{i+1,j,l}}\right)}\phi_{i+1,j,l}$$

This is very similar to what already noted in the 1D cases, in which the interface fluxes are eliminated in favour of node centred fluxes.

The apparent difficulty noted here is that the interface fluxes, actually, still appear in the final formulations as coefficients (see the terms in red). Actually, there is no problem since, in view of an iterative solution, these interface fluxes are updated "at each iteration". So, they can be considered "known" at each step.

So, as usual, we have that interface fluxes are expressed as:

$$\phi_{i+½,j,l} = \alpha_{i,i+1,jl}\, \phi_{i,j,l} + \beta_{i,i+1,jl}\, \phi_{i+1,j,l}$$

It is remarked that *this is the same structure obtained for 1D cases* (see Unit 1), that is now repeated (with more complex coefficients!) in three directions.

By substituting this formulation in the balance equation (*) (see the pages above) we obtain *a classical 7-point formulation, as it is in the case of the simple finite difference or finite volume cases*:

$$\mathcal{O}_{i,j,l}\, \phi_{i,j,l} + \mathcal{W}_{i,j,l}\, \phi_{i-1,j,l} + \mathcal{E}_{i,j,l}\, \phi_{i+1,j,l} + \mathcal{S}_{i,j,l}\, \phi_{i,j-1,l}$$

$$+ \mathcal{N}_{i,j,l}\, \phi_{i,j+1,l} + \mathcal{D}_{i,j,l}\, \phi_{i,j,l-1} + \mathcal{U}_{i,j,l}\, \phi_{i,j,l+1} = 0$$

Obviously enough, the coefficients of this expression are much more complex than the ones obtained by a finite difference scheme. This means that they need more computing time to be calculated.

However, the higher accuracy that can be obtained by the quadratic approximation in the nodes allows the use of a smaller number of nodes with a final advantage.

CUBBOX Method

In order to further improve accuracy, a cubic polynomial can be chosen to represent the trend of neutron flux in each node:

$$\phi(x,y,z) = \phi(\xi,\eta,\zeta) = \phi_{ijl}\,(1 + a_{x,ijl}\,\xi + b_{x,ijl}\,\xi^2 + c_{x,ijl}\,\xi\,(\xi^2 - \tfrac{1}{4})$$

$$+ a_{y,ijl}\,\eta + b_{y,ijl}\,\eta^2 + c_{y,ijl}\,\eta\,(\eta^2 - \tfrac{1}{4})$$

$$+ a_{z,ijl}\,\zeta + b_{z,ijl}\,\zeta^2 + c_{z,ijl}\,\zeta\,(\zeta^2 - \tfrac{1}{4})$$

The particular form of the cubic terms (it is actually a linear + cubic expression), is chosen in a skilled way to simplify calculations. In fact:

- **it does not contribute to the expressions of the coefficients a and b in terms of interfacial fluxes (it is zero at -1/2 and 1/2)**
- **since it is an odd-degree term to be integrated between -1/2 and +1/2, its contribution to the neutron balance is also zero**

So, the relations obtained for QUABOX still hold in this case: very nice and simplifying result!

The further effort to be made in this case is the evaluation of the "c" coefficients. This requires a further use of the Weighted Residuals Method:

$$\int_{V_{ijl}} \left[\text{div } D \text{ grad } \phi + \frac{\nu \Sigma_f}{k} \phi - \Sigma_a \phi \right] w_x \, dV = 0$$

$$\int_{V_{ijl}} \left[\text{div } D \text{ grad } \phi + \frac{\nu \Sigma_f}{k} \phi - \Sigma_a \phi \right] w_y \, dV = 0$$

$$\int_{V_{ijl}} \left[\text{div } D \text{ grad } \phi + \frac{\nu \Sigma_f}{k} \phi - \Sigma_a \phi \right] w_z \, dV = 0$$

The choice in this case is in terms of a "Galerkin" weighting

$$w_x(\xi) = \xi \left(\xi^2 - \frac{1}{4}\right) \qquad w_y(\eta) = \eta \left(\eta^2 - \frac{1}{4}\right)$$

$$w_z(\zeta) = \zeta \left(\zeta^2 - \frac{1}{4}\right)$$

In further works, a similar "collocation" method was selected (Prof. Montagnini and coworkers):

$$w_x(\xi) = \delta\left(\xi - \frac{1}{2}\right) - \delta\left(\xi + \frac{1}{2}\right)$$

$$w_x(\eta) = \delta\left(\eta - \frac{1}{2}\right) - \delta\left(\eta + \frac{1}{2}\right)$$

$$w_x(\zeta) = \delta\left(\zeta - \frac{1}{2}\right) - \delta\left(\zeta + \frac{1}{2}\right)$$

Note that both the above Galerkin and the collocation formulations for the weighting make us of "*odd*" functions

So, we are now in the position to calculate the "c" coefficients, which allow a better accuracy than in the case of quadratic polynomials.

Continuity equations for currents at the interfaces must be then imposed to obtain again a seven point formula in the node centred values of the neutron flux.

We are now in the position to draw some conclusions:

- **"coarse-mesh" methods allow for a *more accurate* evaluation of neutron flux, with the same number of nodes, than finite volume techniques**
- ***the formulations obtained are more complex*, though they finally revert to 3 or 5 or 7 point equations respectively in 1D, 2D and 3D: *this is a characteristics embedded in the different discretization schemes due to the leakage terms***
- **so, the advantage of the greater accuracy is paid by *a larger computational effort per node***
- **the computational effort is larger in the cubic with respect to the quadratic formulation**
- ***however, a lower number of nodes can be used to get still better accuracy that by finite volume techniques: this represents the advantage***

Simplified Example of Eigenvalue Calculations

Foreword

The following material can be used as a basis for *hands-on exercises* to be performed by the students, to make them achieve awareness that *even the simplest techniques discussed during lectures may provide reasonable results*

Of course, more sophisticated techniques will provide more "professional" evaluations: however, simplicity of the analysis is considered an important ingredient to make students "see" that *the basis of complex techniques is already embedded into the simplest ones*

The selected system is *an infinite almost-cylindrical reactor*, with a realistic distribution of assemblies, assumed each one to be internally homogeneous

It must be borne in mind that *the use of the simple one-energy group equation actually makes impossible to accurately evaluate details like the actual effect of reflector or the effectiveness of absorbing elements* (simulating control rods)

However, the treatment was purposely kept "simple" to have a student level treatment *whose results can be sometime checked by simple hand calculations*

In some sense, with due time and some skills in programming, students could have themselves produced both the theory and the software developed for this purpose

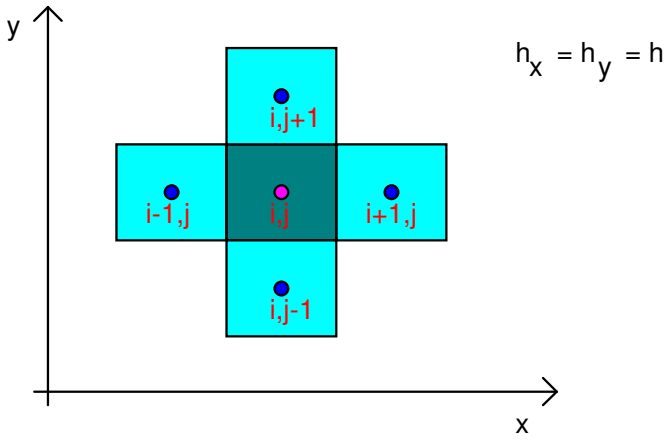
1. Inner iterations

- **Balance equation to be discretised**

$$\text{div } D \text{ grad } \phi - \Sigma_a \phi + S = 0 \qquad (1)$$

Basic choices:

- **square domain in 2D (x and y, infinite z coordinate)**
- **nuclear parameters variable from node to node**
- **simple finite volume discretisation**
- **equal discretisation step along x and y**



The discretised form of (1) can be expressed as:

+ (incoming current) × (related lateral surface)
- (outgoing current) × (related lateral surface)
- (absorption rate) × (node volume)
+ (fixed source) × (node volume) = 0

or

$$-h D_{i,j}\left((\phi_{i,j} - \phi_{i-\frac{1}{2},j})\frac{2}{h} + (\phi_{i,j} - \phi_{i,j-\frac{1}{2}})\frac{2}{h}\right)$$
$$+h D_{i,j}\left((\phi_{i+\frac{1}{2},j} - \phi_{i,j})\frac{2}{h} + (\phi_{i,j+\frac{1}{2}} - \phi_{i,j})\frac{2}{h}\right) \qquad (2)$$
$$-h^2 \Sigma_{a\,i,j}\,\phi_{i.j} + h^2 S_{i,j} = 0$$

$(i = 2, \ldots, N-1) \qquad (j = 2, \ldots, N-1)$

As we know from previous developments in 1D geometry, for any internal node we can eliminate the fluxes at the interfaces by imposing current continuity. For instance

$$-2\frac{D_{i,j}}{h}(\phi_{i+½,j}-\phi_{i,j}) = -2\frac{D_{i+1,j}}{h}(\phi_{i+1,j}-\phi_{i+½,j}) \qquad (3)$$

and then

$$\phi_{i+½,j} = \frac{D_{i,j}\,\phi_{i,j} + D_{i+1,j}\,\phi_{i+1,j}}{D_{i,j}+D_{i+1,j}} \qquad (4)$$

Similar formulations apply for the other three lateral surfaces. Note that in the above definition of the interface flux, the node spacing *has no role* "just because it is equal everywhere" (compare with the 1D case in Unit 1).

It can be easily proven that for internal nodes (2) it is:

$$\mathcal{O}_{i,j}\,\phi_{i,j} = \mathcal{W}_{i,j}\,\phi_{i-1,j} + \mathcal{E}_{i,j}\,\phi_{i+1,j} + \mathcal{S}_{i,j}\,\phi_{i,j-1} + \mathcal{N}_{i,j}\,\phi_{i,j+1} + h^2\,S_{i,j} \qquad (5)$$

where

$$\mathcal{W}_{i,j} = \frac{2\,D_{i,j}\,D_{i-1,j}}{D_{i,j}+D_{i-1,j}} \qquad \mathcal{E}_{i,j} = \frac{2\,D_{i,j}\,D_{i+1,j}}{D_{i,j}+D_{i+1,j}} \qquad (6)$$

$$\mathcal{S}_{i,j} = \frac{2\,D_{i,j}\,D_{i,j-1}}{D_{i,j}+D_{i,j-1}} \qquad \mathcal{N}_{i,j} = \frac{2\,D_{i,j}\,D_{i,j+1}}{D_{i,j}+D_{i,j+1}}$$

In fact, it is for instance:

$$-h\,D_{i,j}\left((\phi_{i,j}-\phi_{i-½,j})\frac{2}{h}\right) = -h\,D_{i,j}\left((\phi_{i,j}-\frac{D_{i,j}\,\phi_{i,j}+D_{i-1,j}\,\phi_{i-1,j}}{D_{i,j}+D_{i-1,j}})\frac{2}{h}\right)$$

$$= -h\,D_{i,j}\left((\frac{D_{i-1,j}\,\phi_{i,j}-D_{i-1,j}\,\phi_{i-1,j}}{D_{i,j}+D_{i-1,j}})\frac{2}{h}\right) = \frac{2\,D_{i,j}\,D_{i-1,j}}{D_{i,j}+D_{i-1,j}}(\phi_{i,j}-\phi_{i-1,j})$$

It can be also noted that it is:

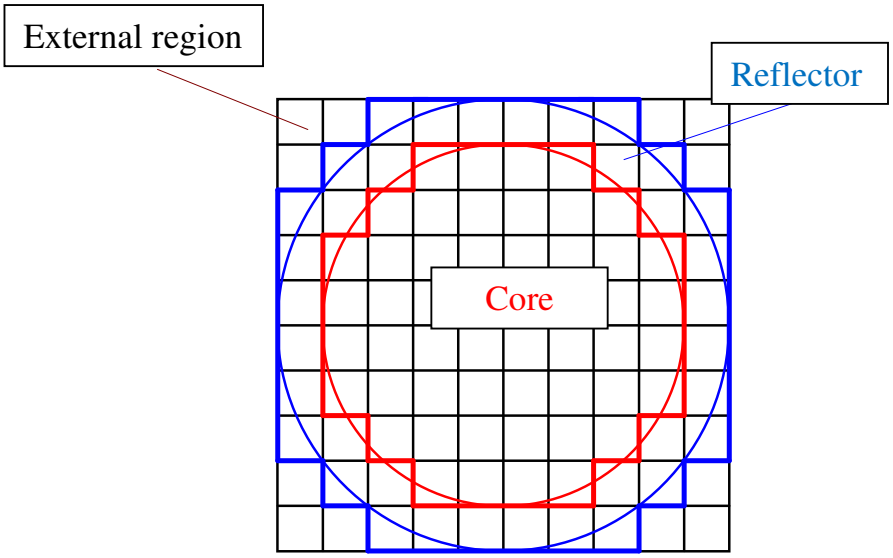
$$\mathcal{O}_{i,j} = \mathcal{W}_{i,j} + \mathcal{E}_{i,j} + \mathcal{S}_{i,j} + \mathcal{N}_{i,j} + h^2 \Sigma_{a\ i,j} \qquad (7)$$

(note, again and again, the *diagonal dominance of the system matrix*).

In "boundary nodes", we have to restart from the balance equation (2) and impose the needed boundary conditions.

For simplicity of treatment, we will employ some trick that should not be adopted in "professional" calculations, since their simplicity is paid by a useless increase in computational effort, which is not justified in real applications.

In particular, though the system to be considered is modelled as a complex boundary on which we should impose the neutron flux to be zero ("extrapolated boundary"), we will impose *the flux to be zero on the boundary of the larger square*. This is simpler and can be obtained assuming that the fluxes in "fictitious nodes" with indices 0 and N+1 is zero.



In particular, the boundary conditions are imposed as:

- $i = 1 \qquad \mathcal{W}_{i,j} = 2\,D_{i,j} \qquad \phi_{0,j} = 0$
- $i = N \qquad \mathcal{E}_{i,j} = 2\,D_{i,j} \qquad \phi_{N+1,j} = 0$

(8)

- $j = 1 \qquad \mathcal{S}_{i,j} = 2\,D_{i,j} \qquad \phi_{i,0} = 0$
- $j = N \qquad \mathcal{N}_{i,j} = 2\,D_{i,j} \qquad \phi_{i,N+1} = 0$

Let's consider, as an example, the case i=1. It is:

$$
\begin{aligned}
&- h\,D_{1,j}\left[(\phi_{1,j} - \phi_{0,j})\frac{2}{h} + (\phi_{1,j} - \phi_{1,j-1/2})\frac{2}{h}\right] \\
&+ h\,D_{1,j}\left[(\phi_{1+1/2,j} - \phi_{i,j})\frac{2}{h} + (\phi_{1,j+1/2} - \phi_{1,j})\frac{2}{h}\right] \\
&- h^2\,\Sigma_{a\,1,j}\,\phi_{1,j} + h^2\,S_{1,j} = 0
\end{aligned}
\qquad (9)
$$

$(j = 2, \ldots, N-1)$

It is noted that, after we have replaced the interface node with $\phi_{0,j} = 0$ the "west" coefficient becomes $\mathcal{W}_{i,j} = 2\,D_{i,j}$.

The way of imposing the "really wanted" boundary conditions around the periphery of the reflector is specified hereafter.

Reactor Model Description

- **As seen in the previous sketch, nuclear parameters are assigned for "core" nodes, and "reflector" nodes**
- **It is also possible to redefine single node parameters, e.g. for simulating localized absorption (control rods)**

- **The trick adopted to assign the condition of zero flux on the outer surface of the reflector on the basis of the zero flux assigned on the external contour of the square region consists in *assigning a very large value of the diffusion coefficient in the "external region"*; in fact:**
  - **whatever the leakage current, *a large diffusion coefficient means a low neutron flux gradient***
  - ***the low neutron flux gradient, in turn, makes the flux on the external surface of the reflector to be very close to the one on the outer surface of the square region (assigned to be zero)***
- **In addition the absorption cross section in the external region is also assigned to be large**
- **This numerical trick allows:**
  - ***simplifying the input deck and the solution algorithm,* operating on a simple square domain**
  - ***maintain the diagonal dominance of the system matrix* (the absorption cross section is large)**
- **On the other hand, the disadvantage of this technique is to calculate "useless nodes", in which the neutron flux is anyway zero:**

  an optimized program should not work in this way

Adopted linear system solution methods:

1. Jacobi

$$\mathcal{O}_{i,j}\,\phi_{i,j}^{(m+1)} = \boxed{\mathcal{W}_{i,j}\,\phi_{i-1,j}^{(m)} + \mathcal{E}_{i,j}\,\phi_{i+1,j}^{(m)} + \mathcal{S}_{i,j}\,\phi_{i,j-1}^{(m)} + \mathcal{N}_{i,j}\,\phi_{i,j+1}^{(m)}} + h^2\,S_{i,j} \qquad (10)$$

2. Gauss-Seidel

$$\mathcal{O}_{i,j}\,\phi_{i,j}^{(m+1)} = \mathcal{W}_{i,j}\,\phi_{i-1,j}^{(m+1)} + \mathcal{E}_{i,j}\,\phi_{i+1,j}^{(m)} + \mathcal{S}_{i,j}\,\phi_{i,j-1}^{(m+1)} + \mathcal{N}_{i,j}\,\phi_{i,j+1}^{(m)} + h^2\,S_{i,j} \tag{11}$$

3. SOR

$$\phi_{i,j}^{(m+1)} = \omega\left(\phi_{i,j}^{(m+1)} - \phi_{i,j}^{(m)}\right)_{GS} + \phi_{i,j}^{(m)} \tag{12}$$

4. LOR

$$\begin{aligned} -\mathcal{W}_{i,j}\,\phi_{i-1,j}^{(m+1)} + \mathcal{O}_{i,j}\,\phi_{i,j}^{(m+1)} - \mathcal{E}_{i,j}\,\phi_{i+1,j}^{(m+1)} = \\ +\omega\left(\mathcal{S}_{i,j}\,\phi_{i,j-1}^{(m+1)} + \mathcal{N}_{i,j}\,\phi_{i,j+1}^{(m)} + h^2\,S_{i,j}\right) \\ +(1-\omega)\left(-\mathcal{W}_{i,j}\,\phi_{i-1,j}^{(m)} + \mathcal{O}_{i,j}\,\phi_{i,j}^{(m)} - \mathcal{E}_{i,j}\,\phi_{i+1,j}^{(m)}\right) \end{aligned} \tag{13}$$

- **Outer Iterations**

The power method is used. We put:

$$\underline{\underline{M}} = \underset{i}{\text{diag}}\,(\nu\Sigma_f)_i \tag{14}$$

and then

$$\underline{\underline{A}}\,\underline{\phi} = \frac{1}{k}\,\underline{\underline{M}}\,\underline{\phi} = \frac{1}{k}\,\underline{\psi} \tag{15}$$

obtaining

$$\underline{\underline{K}}\,\underline{\phi} = k\,\underline{\phi} \tag{16}$$

where

$$\underline{\underline{K}} = \underline{\underline{A}}^{-1}\,\underline{\underline{M}} \tag{17}$$

The iterative process has the "theoretical" form:

$$\underline{\phi}^{(n+1)} = \frac{1}{k^{(n)}} \underline{\underline{K}}\, \underline{\phi}^{(n)} \qquad (18)$$

and it is

$$k^{(n+1)} = k^{(n)} \frac{\sum_{i=1}^{N} \psi_i^{(n+1)}}{\sum_{i=1}^{N} \psi_i^{(n)}} \qquad (19)$$

Actually, the solution procedure is the following:

- **diffusion equations are solved with the chosen iterative scheme starting with a guessed source**
- **making use of the new fluxes the source is updated**
- **the value of k is therefore updated**
- **iterations are performed until convergence**

FORTRAN Programme

```
c-----------------------------------------------------------------------c
c                                                                       c
c                         Programma   T P R S                           c
c                                                                       c
c                  Teaching Purpose Reactor Simulation                  c
c                                                                       c
c                      W. Ambrosini, Ottobre 1997                       c
c                                                                       c
c-----------------------------------------------------------------------c
      program core
      implicit double precision (a-h,o-z)
      character*80 riga
c
      parameter (m = 101)
      common /generl/ h,aleng,rcore,rrefl,omega,n,method,istvid
      common /nuclea/ diff(m,m),sigma(m,m),anisif(m,m),source(m,m)
      common /flux/ phi(0:m,0:m),oldphi(0:m,0:m),akeff
      common /winds/ ce(m,m),cw(m,m),cn(m,m),cs(m,m),co(m,m)
      dimension oldsou(m,m),class(m,m)
c
      open (unit=5,file='core.dat')
      open (unit=6,file='core.out')
      open (unit=7,file='core.pla')
      open (unit=8,file='core.txt')
c
c   lettura dei dati generali
c
      read(5,100) riga
      read(5,*) aleng,rcore,rrefl,n,method,omega
c
c   primo processamento per ottenere costanti di interesse
c   (first pre-processing for preparing relevant constants)
c
      h = aleng / dfloat (n)
      halfal = 0.5d00 * aleng
      halfh = 0.5d00 * h
      pi = 4.d00 * datan (1.d00)
c
c   lettura delle costanti nucleari del nocciolo
c   (reading the nuclear constants of the reactor core)
c
      read(5,100) riga
      read(5,*) dcore,sigcor,anisic
c
c   lettura delle costanti nucleari del riflettore
c   (option for printing on the video)
c
      read(5,100)
      read(5,*) drefl,sigref
c
c   opzione di stampa a video
c   (option for printing on the video)
c
      write(*,*) ' Video printing ? (1=yes; 0=no) '
      read(*,*) istvid
c
c   assegnazione delle costanti nucleari nei nodi
c   (assigning nuclear constants in the nodes)
c
      sumsou = 0.d00
      do 10 i = 1,n
      xi = h * dfloat(i) - halfh
      deltxi = xi - halfal
```

```
c
      do 10 j = 1,n
      yj = h * dfloat(j) - halfh
      deltyj = yj - halfal
c
      radius = dsqrt ( deltxi * deltxi + deltyj * deltyj )
c
c  nodi del nocciolo
c  (core nodes)
        if(radius.le.rcore) then
        diff(i,j) = dcore
        sigma(i,j) = sigcor
        anisif(i,j) = anisic
        ratr = radius / rcore
c
        soucor = 1.84d13
        source(i,j) = soucor * dcos ( 0.5d00 * pi * ratr )
        sumsou = sumsou + source(i,j)
        oldsou(i,j) = source(i,j)
        class(i,j) = 2.d00
c
c  nodi del riflettore
c  (reflector nodes)
        elseif(radius.le.rrefl) then
        diff(i,j) = drefl
        sigma(i,j) = sigref
        source(i,j) = 0.d00
        class(i,j) = 1.d00
c
c  nodi esterni
c  (external nodes)
        else
        diff(i,j) = 1.d05
        sigma(i,j) = 1.d05
        source(i,j) = 0.d00
        class(i,j) = 0.d00
        endif
c
   10 continue
c
c   assegnazione delle costanti in nodi particolari (npart > 0)
c   (assigning constants in particular nodes (if npart > 0)
c
      read(5,100) riga
      read(5,*) npart
c
        if(npart.gt.0) then
        read(5,100) riga
        do 20 ipart = 1,npart
        read(5,*) ip,jp,diff(ip,jp),sigma(ip,jp),anisif(ip,jp)
        xip = h * dfloat(ip) - halfh
        yjp = h * dfloat(jp) - halfh
        class(ip,jp) = 3.d00
   20   continue
        endif
c
c   costruzione dei coefficienti dell'equazione della diffusione
c   (the coefficients of the diffusion equation are set up)
c
      do 30 i = 1,n
      do 30 j = 1,n
c
        if(i.eq.1) then
        cw(i,j) = 2.d00 * diff(i,j)
        else
        cw(i,j) = 2.d00 * diff(i,j) * diff(i-1,j)
```

```
     &                   / ( diff(i,j) + diff(i-1,j) )
          endif
c
          if(i.eq.n) then
          ce(i,j) = 2.d00 * diff(i,j)
          else
          ce(i,j) = 2.d00 * diff(i,j) * diff(i+1,j)
     &                   / ( diff(i,j) + diff(i+1,j) )
          endif
c
          if(j.eq.1) then
          cs(i,j) = 2.d00 * diff(i,j)
          else
          cs(i,j) = 2.d00 * diff(i,j) * diff(i,j-1)
     &                   / ( diff(i,j) + diff(i,j-1) )
          endif
c
          if(j.eq.n) then
          cn(i,j) = 2.d00 * diff(i,j)
          else
          cn(i,j) = 2.d00 * diff(i,j) * diff(i,j+1)
     &                   / ( diff(i,j) + diff(i,j+1) )
          endif
c
       co(i,j) = ce(i,j) + cw(i,j) + cn(i,j) + cs(i,j)
     &                   + h * h * sigma(i,j)
c
   30 continue
c
c    iterazioni esterne
c    (external iterations)
c
        call gettim(ihr,imin,isec,icent)
        time0 = 3600.d00 * ihr + 60.d00 * imin + isec + 0.01d00 * icent
c
        akeff = 1.d00
        do 50 itext = 1,100000
c
c    soluzione del sistema con il metodo prescelto
c    (algebraic system solution with the selected method)
c
        if(method.eq.1) call jacobi (iter,rspect,itext)
        if(method.eq.2) call gseid (iter,itext)
        if(method.eq.3) call sor (iter,itext)
        if(method.eq.4) call lor (iter,itext)
c
        oldak = akeff
        oldsum = sumsou
c
        sumsou = 0.d00
        do 40 i = 1,n
        do 40 j = 1,n
        source(i,j) = anisif(i,j) * phi(i,j)
        sumsou = sumsou + source(i,j)
   40 continue
c
        akeff = oldak * sumsou / oldsum
c
           if(istvid.eq.1) then
           write(*,120) itext,akeff
           endif
        write(6,120) itext,akeff
c
        if(dabs(akeff-oldak).lt.1.d-7) goto 55
   50 continue
c
```

```
c
   55 continue
c
      call gettim(ihr,imin,isec,icent)
      time1 = 3600.d00 * ihr + 60.d00 * imin + isec + 0.01d00 * icent
      tcpu = time1 - time0
c
          if(method.eq.1) then
          omeopt = 2.d00 / ( 1.d00 + dsqrt ( 1.d00 - rspect * rspect ) )
               if(istvid.eq.1) then
               write(*,130) rspect
               write(*,135) omeopt
               endif
          write(6,130) rspect
          write(6,135) omeopt
          endif
c
c   scrittura dei valori del flusso
c
      area = 0.d00
      totpow = 0.d00
      powmax = 0.d00
      phimed = 0.d00
      sumasf = 0.d00
      do 60 i = 1,n
      xi = h * dfloat(i) - halfh
      do 60 j = 1,n
      yj = h * dfloat(j) - halfh
c
      sumasf = sumasf + anisif(i,j)
      phiasf = phi(i,j) * anisif(i,j)
      pow = phiasf / ( 2.5d00 * 3.1e10 )
      phimed = phimed + phiasf
      totpow = totpow + pow * h * h * 365.76d00
      if(powmax.lt.pow) powmax = pow
c
      write(7,160) xi,yj,class(i,j)
   60 write(8,110) xi,yj,phi(i,j)
      phimed = phimed / sumasf
      powavg = totpow / ( pi * rcore * rcore * 365.76d00 )
      fattpc = powmax / powavg
      react = ( akeff - 1.d00 ) / akeff
c
         if(istvid.eq.1) then
         write(*,170) react
         write(*,180) totpow
         write(*,190) phimed
         write(*,140) fattpc
         write(*,150) tcpu
         endif
c
      write(6,121) itext,akeff
      write(6,170) react
      write(6,180) totpow
      write(6,190) phimed
      write(6,140) fattpc
      write(6,150) tcpu
c
      stop
  100 format (a80)
  110 format (3(1x,e14.7))
  120 format (/,1x,' External Iteration n. ',i5,' Keff = ',f12.9)
  121 format (/,1x,' External Iteration number = ',i5,' Keff = ',f12.9)
  130 format (/,1x,' Jacobi matrix spectral radius = ',f12.9)
  135 format (1x,' Optimal Overrelaxation Parameter = ',f12.9)
```

```
  140 format (1x,' Radial Peaking Factor = ',f12.9)
  150 format (/,1x,' Processing time = ',f7.2,' s ')
  160 format (3(1x,e14.7))
  170 format (/,1x,' Reactivity = ',f12.9)
  180 format (1x,' Total Thermal Power = ',1pe14.7,' W ')
  190 format (1x,' Average Flux = ',1pe14.7,' n/(cm2.s)')
      end
c-----------------------------------------------------------------------c
c                                                                       c
c     Soluzione delle Equazioni della Diffusione con Jacobi             c
c     (Solving the diffusion equations with the Jacobi method)          c
c                                                                       c
c-----------------------------------------------------------------------c
      subroutine jacobi (iter,rspect,itext)
      implicit double precision (a-h,o-z)
c
      parameter (m = 101)
      common /generl/ h,aleng,rcore,rrefl,omega,n,method,istvid
      common /nuclea/ diff(m,m),sigma(m,m),anisif(m,m),source(m,m)
      common /flux/ phi(0:m,0:m),oldphi(0:m,0:m),akeff
      common /winds/ ce(m,m),cw(m,m),cn(m,m),cs(m,m),co(m,m)
c
c   assegnazione dell'approssimazione iniziale
c   (the initial approximation is assigned)
c
      np1 = n + 1
      h2 = h * h
      do 10 i = 0,np1
      do 10 j = 0,np1
c
       if(itext.eq.1) then
         if( (i.eq.0).or.(j.eq.0).or.(i.eq.np1).or.(j.eq.np1) ) then
         phi(i,j) = 0.d00
         oldphi(i,j) = 0.d00
         else
         phi(i,j) = h2 * source(i,j) / co(i,j) / akeff
         oldphi(i,j) = phi(i,j)
         endif
       else
       oldphi(i,j) = phi(i,j)
       endif
c
   10 continue
c
      oldel2 = 1.d00
      do 30 iter = 1,10000000
c
      ratmax = 0.d00
      del2su = 0.d00
c
      do 20 j = 1,n
      do 20 i = 1,n
      tm = ce(i,j) * oldphi(i+1,j) + cn(i,j) * oldphi(i,j+1)
     &   + cw(i,j) * oldphi(i-1,j) + cs(i,j) * oldphi(i,j-1)
c
      phi(i,j) = ( tm + h2 * source(i,j) / akeff ) / co(i,j)
c
      delphi = phi(i,j) - oldphi(i,j)
      del2su = del2su + delphi * delphi
c     ratio = dabs (delphi) / ( phi(i,j) + 1.d-10 )
      ratio = dabs (delphi)
      if(ratio.gt.ratmax) ratmax = ratio
c
   20 continue
c
      do 25 i = 1,n
```

```
      do 25 j = 1,n
      oldphi(i,j) = phi(i,j)
   25 continue
c
      rspect = dsqrt ( del2su / oldel2 )
      oldel2 = del2su
c
      if((ratmax.lt.1.d00).and.(iter.ne.1)) goto 35
c
      if(istvid.eq.1) write(*,101) iter,ratmax
      write(6,101) iter,ratmax
   30 continue
      write(6,100)
c
   35 return
  100 format (/,1x,'<<<<< Warning: Convergence Problems >>>>>')
  101 format (' Jacobi Internal Iteration n. ',i6,
     & ' Max. Error = ',e14.7)
      end
c-----------------------------------------------------------------------c
c                                                                       c
c     Soluzione delle Equazioni della Diffusione con Gauss-Seidel       c
c     (Solving the diffusion equations with the Gauss-Seidel method)    c
c                                                                       c
c-----------------------------------------------------------------------c
      subroutine gseid (iter,itext)
      implicit double precision (a-h,o-z)
c
      parameter (m = 101)
      common /generl/ h,aleng,rcore,rrefl,omega,n,method,istvid
      common /nuclea/ diff(m,m),sigma(m,m),anisif(m,m),source(m,m)
      common /flux/ phi(0:m,0:m),oldphi(0:m,0:m),akeff
      common /winds/ ce(m,m),cw(m,m),cn(m,m),cs(m,m),co(m,m)
c
c   assegnazione dell'approssimazione iniziale
c   (the initial approximation is assigned)
c
      np1 = n + 1
      h2 = h * h
      do 10 i = 0,np1
      do 10 j = 0,np1
c
       if(itext.eq.1) then
         if( (i.eq.0).or.(j.eq.0).or.(i.eq.np1).or.(j.eq.np1) ) then
         phi(i,j) = 0.d00
         oldphi(i,j) = 0.d00
         else
         phi(i,j) = h2 * source(i,j) / co(i,j) / akeff
         oldphi(i,j) = phi(i,j)
         endif
       else
       oldphi(i,j) = phi(i,j)
       endif
c
   10 continue
c
      do 30 iter = 1,10000000
c
      ratmax = 0.d00
c
      do 20 j = 1,n
      do 20 i = 1,n
      tm = ce(i,j) * phi(i+1,j) + cn(i,j) * phi(i,j+1)
      tmp1 = cw(i,j) * phi(i-1,j) + cs(i,j) * phi(i,j-1)
c
      phi(i,j) = ( tm + tmp1 + h2 * source(i,j) / akeff ) / co(i,j)
```

```
c
      delphi = phi(i,j) - oldphi(i,j)
c     ratio = dabs (delphi) / ( phi(i,j) + 1.d-10 )
      ratio = dabs (delphi)
      if(ratio.gt.ratmax) ratmax = ratio
c
      oldphi(i,j) = phi(i,j)
c
   20 continue
c
      if((ratmax.lt.1.d00).and.(iter.ne.1)) goto 35
c
      if(istvid.eq.1) write(*,101) iter,ratmax
      write(6,101) iter,ratmax
   30 continue
      write(6,100)
c
   35 return
  100 format (/,1x,'<<<<< Warning: Convergence Problems >>>>>')
  101 format (' GS Internal Iteration n. ',i6,
     & ' Max. Error = ',e14.7)
      end
c-----------------------------------------------------------------------c
c                                                                       c
c     Soluzione delle Equazioni della Diffusione con SOR                c
c     (Solving the diffusion equations with the SOR method)             c
c                                                                       c
c-----------------------------------------------------------------------c
      subroutine sor (iter,itext)
      implicit double precision (a-h,o-z)
c
      parameter (m = 101)
      common /generl/ h,aleng,rcore,rrefl,omega,n,method,istvid
      common /nuclea/ diff(m,m),sigma(m,m),anisif(m,m),source(m,m)
      common /flux/ phi(0:m,0:m),oldphi(0:m,0:m),akeff
      common /winds/ ce(m,m),cw(m,m),cn(m,m),cs(m,m),co(m,m)
c
c   assegnazione dell'approssimazione iniziale
c   (the initial approximation is assigned)
c
      np1 = n + 1
      h2 = h * h
      do 10 i = 0,np1
      do 10 j = 0,np1
c
       if(itext.eq.1) then
         if( (i.eq.0).or.(j.eq.0).or.(i.eq.np1).or.(j.eq.np1) ) then
         phi(i,j) = 0.d00
         oldphi(i,j) = 0.d00
         else
         phi(i,j) = h2 * source(i,j) / co(i,j) / akeff
         oldphi(i,j) = phi(i,j)
         endif
       else
       oldphi(i,j) = phi(i,j)
       endif
c
   10 continue
c
      do 30 iter = 1,10000000
c
      ratmax = 0.d00
c
      do 20 j = 1,n
      do 20 i = 1,n
      tm = ce(i,j) * phi(i+1,j) + cn(i,j) * phi(i,j+1)
```

```
      tmp1 = cw(i,j) * phi(i-1,j) + cs(i,j) * phi(i,j-1)
c
      phi(i,j) = ( tm + tmp1 + h2 * source(i,j) / akeff ) / co(i,j)
c
c   sovrarilassamento
c   (overrelaxation)
c
      phi(i,j) = ( phi(i,j) - oldphi(i,j) ) * omega + oldphi(i,j)
c
      delphi = phi(i,j) - oldphi(i,j)
c     ratio = dabs (delphi) / ( phi(i,j) + 1.d-10 )
      ratio = dabs (delphi)
      if(ratio.gt.ratmax) ratmax = ratio
c
      oldphi(i,j) = phi(i,j)
c
   20 continue
c
      if((ratmax.lt.1.d00).and.(iter.ne.1)) goto 35
c
      if(istvid.eq.1) write(*,101) iter,ratmax
      write(6,101) iter,ratmax
   30 continue
      write(6,100)
c
   35 return
  100 format (/,1x,'<<<<< Warning: Convergence Problems >>>>>')
  101 format (' SOR Internal Iteration n. ',i6,
     & ' Max. Error = ',e14.7)
      end
c-----------------------------------------------------------------------c
c                                                                       c
c     Soluzione delle Equazioni della Diffusione con LOR                c
c     (Solving the diffusion equations with the LOR method)             c
c                                                                       c
c-----------------------------------------------------------------------c
      subroutine lor (iter,itext)
      implicit double precision (a-h,o-z)
c
      parameter (m = 101)
      common /generl/ h,aleng,rcore,rrefl,omega,n,method,istvid
      common /nuclea/ diff(m,m),sigma(m,m),anisif(m,m),source(m,m)
      common /flux/ phi(0:m,0:m),oldphi(0:m,0:m),akeff
      common /winds/ ce(m,m),cw(m,m),cn(m,m),cs(m,m),co(m,m)
c
      dimension a(m),b(m),c(m),d(m),v(m),alef(m),bet(m)
c
c   assegnazione dell'approssimazione iniziale
c   (the initial approximation is assigned)
c
      np1 = n + 1
      h2 = h * h
      do 10 i = 0,np1
      do 10 j = 0,np1
c
       if(itext.eq.1) then
         if( (i.eq.0).or.(j.eq.0).or.(i.eq.np1).or.(j.eq.np1) ) then
         phi(i,j) = 0.d00
         oldphi(i,j) = 0.d00
         else
         phi(i,j) = h2 * source(i,j) / co(i,j) / akeff
         oldphi(i,j) = phi(i,j)
         endif
       else
       oldphi(i,j) = phi(i,j)
       endif
```

```
c
   10 continue
c
      do 40 iter = 1,10000000
c
      ratmax = 0.d00
c
c  ciclo sull'indice che definisce le linee orizzontali
c  (loop on the index defining the horizontal lines)
c
      do 30 j = 1,n
c
c  assegnazione dei coefficienti della matrice tridiagonale di linea
c  (assigning the coefficients of the tridiagonal matrix for the line)
c
      do 20 i = 1,n
      a(i) = - cw(i,j)
      b(i) = co(i,j)
      c(i) = - ce(i,j)
      d(i) = omega * ( h2 * source(i,j) / akeff
     &                 + cs(i,j) * phi(i,j-1) + cn(i,j) * phi(i,j+1) )
     &  + (1.d00 - omega) * ( - cw(i,j) * phi(i-1,j)
     &            + co(i,j) * phi(i,j) - ce(i,j) * phi(i+1,j) )
   20 continue
c
c  soluzione del sistema tridiagonale
c  (solution of the tri-diagonal system)
c
      call tdma (a,b,c,d,v,alef,bet,n,m)
c
c  assegnazione dei valori del flusso sulla linea j = costante
c  e valutazione dell'errore massimo
c  (assigning the values of the fluxes on the line at j = constant
c  and maximum error evaluation)
c
      do 25 i = 1,n
      oldphi(i,j) = phi(i,j)
      phi(i,j) = v(i)
c
      delphi = phi(i,j) - oldphi(i,j)
c     ratio = dabs (delphi) / ( phi(i,j) + 1.d-10 )
      ratio = dabs (delphi)
      if(ratio.gt.ratmax) ratmax = ratio
   25 continue
c
   30 continue
c
      if((ratmax.lt.1.d00).and.(iter.ne.1)) goto 45
c
      if(istvid.eq.1) write(*,101) iter,ratmax
      write(6,101) iter,ratmax
   40 continue
      write(6,100)
c
   45 return
  100 format (/,1x,'<<<<< Warning: Convergence Problems >>>>>')
  101 format (' LOR Internal Iteration n. ',i6,
     & ' Max. Error = ',e14.7)
      end
```

```
c-----------------------------------------------------------------------c
c                                                                       c
c   Soluzione di Sistemi Tridiagonali con l'Algoritmo di Thomas         c
c   (Solution of systems with Tridiagonal matrix                        c
c    by the Thomas algorithm)                                           c
c                                                                       c
c-----------------------------------------------------------------------c
      subroutine tdma (a,b,c,d,v,alef,bet,n,ld)
      implicit double precision (a-h,o-z)
      dimension a(ld),b(ld),c(ld),d(ld),v(ld),alef(ld),bet(ld)
      ub=1.d00/b(1)
      alef(1)=c(1)*ub
      bet(1)=d(1)*ub
      do 10 i=2,n
      l=i-1
      qz=b(i)-a(i)*alef(l)
      uqz=1.d00/qz
      alef(i)=c(i)*uqz
   10 bet(i)=(d(i)-a(i)*bet(l))*uqz
      nm1=n-1
      v(n)=bet(n)
      do 20 i=1,nm1
      ii=n-i
      l=ii+1
   20 v(ii)=bet(ii)-alef(ii)*v(l)
      return
      end
```

Proposed activity: read the programme trying to understand its working principles. A minimum knowledge of programming structures is needed.

1. **Consider the way in which nuclear constants are assigned**
2. **Consider the way in which "east", "west", "north" and "south" coefficients are assigned considering the boundary conditions**
3. **Identify the loop for "outer iterations" and the computation of K_{eff} with the "generational formulation"**
4. **Identify the loop for "inner iterations"**
5. **Consider the "slight" difference in programming Jacobi and Gauss-Seidel methods**
6. **Consider the difference between Gauss-Seidel and SOR**
7. **Consider the structure of programming LOR by lines and understand what possible changes should be made for programing the same scheme by columns (THIS WILL CLARIFY SOME OF THE DOUBTS RAISED DURING LECTURES)**

APPLICATION EXAMPLES

1. Bare Cylindrical Reactor (file: nudo.dat)

Reference data

Square domain with a side of 300 cm

Square cells with a side of 10 cm

Core radius = 125 cm

Analytical solution:

- The neutron flux distribution is: $\phi(r) = \phi_0 J_0\left(\frac{2.4048\ r}{R}\right)$

The radial peaking factor is:

$$F_\perp = \frac{\phi_{max}}{\phi_{med}} = \frac{\phi_0}{\frac{\phi_0}{\pi R^2}\int_0^R J_0\left(\frac{2.4048\ r}{R}\right) 2\pi\, r\, dr} \approx 2.316...$$

where $B^2 = \left(\frac{2.4048}{R}\right)^2$

- Assuming:

$\Sigma_a = 0.08\ cm^{-1}$ $\quad \nu\Sigma_f = 0.0807\ cm^{-1}$ $\quad D = 0.4$ cm

it is expected

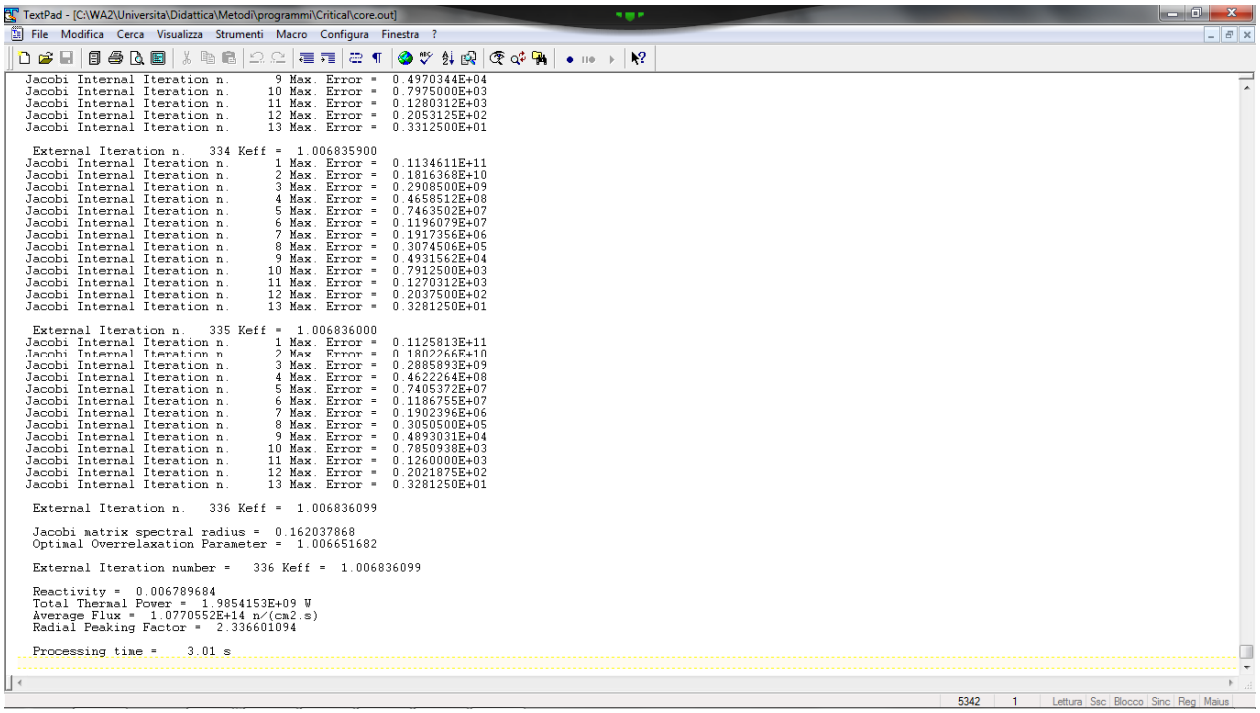
$$K_{eff} = \frac{\nu\Sigma_f}{\Sigma_a + D\,B^2} \approx 1.006887$$

Input deck:

```
aleng     rcore     rrefl         n    method     omega
 300.      125.      125.        30         1        1.
dcore    sigcor    anisic
  0.4      0.08    0.0807
drefl    sigref
  0.3     0.010
npart
    0
```

Run this case by the following procedure:

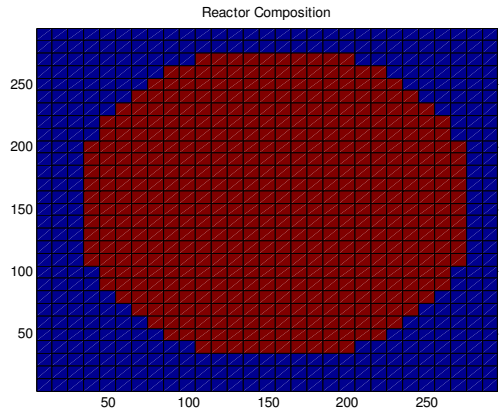
- **copy by a text editor the content of "nudo.dat" into the file "core.dat" and save**
- **click on Critical-32.exe or Critical-64.exe for 32 bit or 64 bit processors)**
- **select "1" on the screen if you want displaying the iterations**
- **open the "core.out" file by a text editor and look at the results; for this case you have a long file whose last part is:**

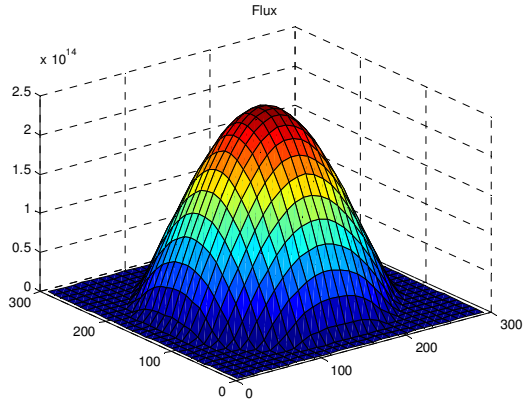


It can be noted that:

- **the Keff is close to the analytical prediction for a "cylindircal reactor" (ask yourself if the present one is really cylindrical…)**
- **the radial peaking factor is also close to the theoretical prediction**
- **the spectral radius of the Jacobi iteration matrix is quite low**
- **the thermal power was assigned arbitrarily: never mind…**

- **on your PC you will certainly find different computing times (maybe different from time to time…)**
- **you can see how many "outer iterations" are needed for this case without proper acceleration**
- **open the MATLAB file "critical.m" and run it to visualize the core description and the flux distribution (click on the first figure to see the second one)**

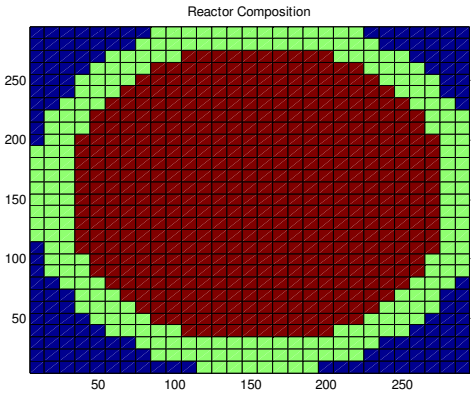


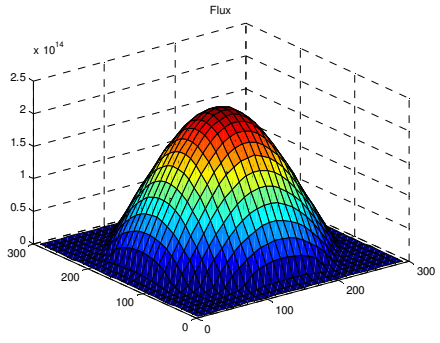


- **run now with different values of "method" (2 to 4) for GS, SOR, LOR you will find not so much difference in the number of inner iterations because in this case the spectral radius of the Jacobi matrix is already low; potspone judgement on this feature…**

2. Cylindrical reactor with reflector (file: reflect.dat)

aleng	rcore	rrefl	n	method	omega
300.	125.	150.	30	1	1.0
dcore	sigcor	anisic			
0.4	0.08	0.0807			
drefl	sigref				
0.3	0.010				
npart					
0					

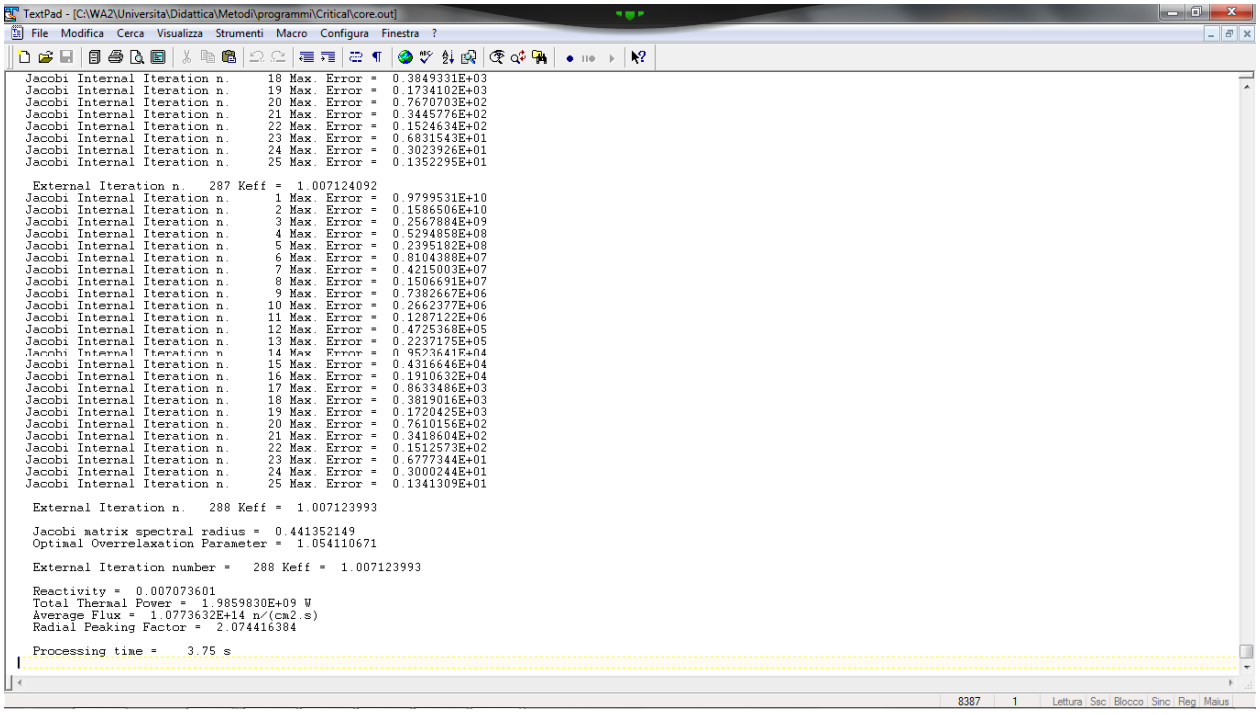




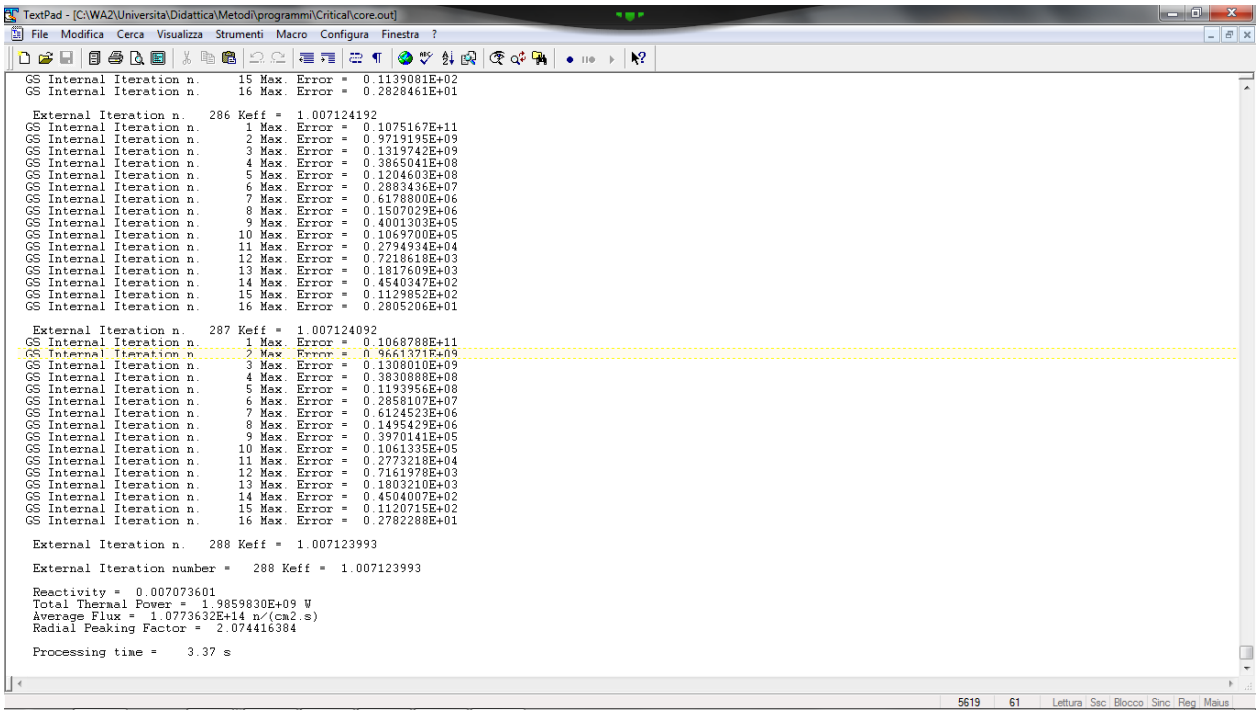
NB: There is a bug in the Matlab file that I was not able to identify. It is cutting one line and one column in the visualization (see above). Sorry ! Whoever solves it will have 0.1/30 more on the final vote of his examinations ;-) .

Repeat the same analyses with the reflector case (again, copy the content of the input file into the core.dat file); you will find:

- a greater value of the Keff: why?
- a slightly different nrutron flux distribution (not so much);
- an increased number of inner iterations:
- e.g., with Jacobi you will find the following:



Selecting the GS method it is found:



i.e. the same Keff, but with a lower number of inner iterations: not halved as asymptotically expected… why? try answering…

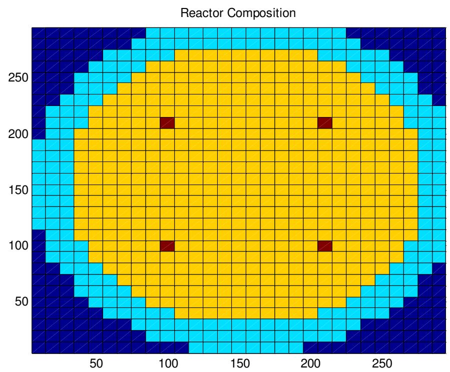
Obviously it is because now the spectral radius of the Jacobi matrix has increased (look in the output of the Jacobi case) because of less average absorption; shifting to GS is now more effective. Try now with SOR and the suggested optimal ω parameter.

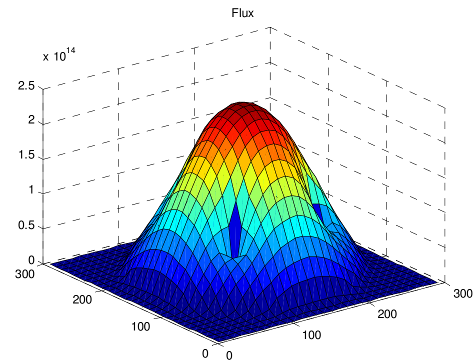
Further suggested activity: Play with the radius of the core restricting it very much (small fissile region) in order to see increasing the spectral radius of the Jacobi matrix. Use now the different numerical schemes to see the difference: Jacobi vs. GS vs. SOR vs. LOR.

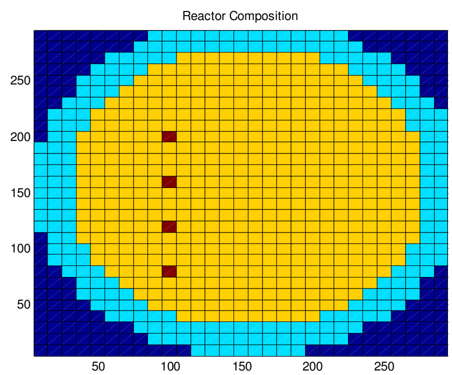
3. Cylindrical reactor with reflector and four control rods

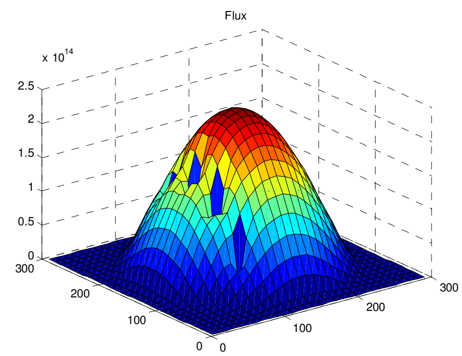
- **at the vertices of a square (file: 4rodssqu.dat)**
- **at the reactor core centre (file: 4rodsmid.dat)**
- **placed laterally (file: 4rodslat.dat)**

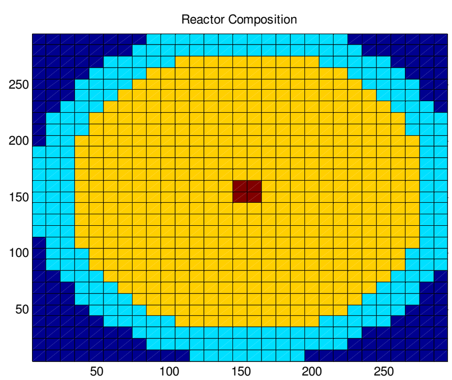
```
aleng      rcore      rrefl       n     method     omega
 300.       125.       150.       30        4         1.
dcore      sigcor     anisic
  0.4        0.08     0.0807
drefl      sigref
  0.3       0.010
npart
    4
   ip         jp       diff     sigma    anisic
   10         10        0.4      0.13    0.0807
   10         21        0.4      0.13    0.0807
   21         10        0.4      0.13    0.0807
   21         21        0.4      0.13    0.0807
```

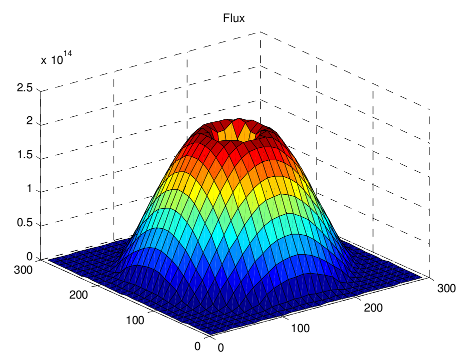








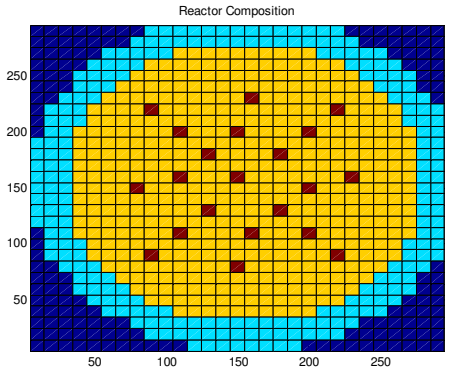


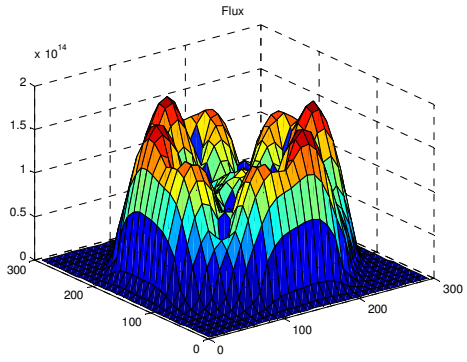


- **Compare the different configurations of the control rods an draw conclusions about their worth.**

- **Play with the different numerical solution schemes and draw conclusions**

4. Cylindrical reactor with reflector and many control rods (file: manyrods.dat)





- **Play with the different numerical solution schemes and draw conclusions**

Finally, discuss with your teacher the limited realism obtained by the use of a single energy group equations

FURTHER PROPOSED ACTIVITIES

1. **Study the effect of a single control rod placed in different locations in the reactor**

   Hint:

   make plots of $\Delta\rho$ as a function of the local squared neutron flux

2. **Study the best positioning of four control rods for getting the minimum K_{eff}**

3. **Optimise the positioning of the control rod pattern starting with the file manyrods.dat**