# “Island Sensitivity Code” documentation

Alessandro Geraldini

March 3, 2021

*This documentation is still incomplete.*

## 1 Purpose

The **Island Sensitivity Code (ISC)** calculates the island width and Residue[1] (two different measures of size of a magnetic island) and the gradients of these quantities with respect to any magnetic field parameter. The only thing needed is a magnetic field function, along with first and second spatial derivatives in the poloidal plane, and the magnetic field gradient function, which calculates the gradient of the magnetic field and its first spatial derivative in the poloidal plane with respect to any of the magnetic field parameters.

## 2 Outline of scripts and functions

In its current form ISC is written in C and is composed of the set of scripts:

- **analyze.c**: **main()** function for analyzing a magnetic configuration and its islands.
- **optimize.c**: **main()** function for optimizing a magnetic configuration and its islands. At the moment it is very basic and can handle only simple optimizations, thus it needs to be heavily updated or replaced for optimizations over more parameters.
- **magfield.c**: several functions are included here
  - **fetchparams()**: takes no arguments; returns a structure of type **fieldparams** containing all the information necessary to evaluate the magnetic field. Currenly three magnetic field types are treated: **Reim, heli, coil** which stand for Reiman, helical, coils.
  - **BReim(. . . )**: takes 9 arguments, three of which specify the position at which the magnetic field is evaluated and the rest specify the parameters of the magnetic field of type **Reim**; returns a structure of type **field** containing the magnetic field, its first and second derivatives in the poloidal plane.
  - **Bcoil(. . . )**: takes 6 arguments, 3 specify the position at which the magnetic field is evaluated and the remaining 3 specify the coils producing the magnetic field of type **coil**; returns a structure of type **field** containing the magnetic field, its first and second derivatives in the poloidal plane specified by the toroidal angle.

[1] The residue and its gradient can be calculated for any periodic magnetic field line, even when it is not an O point/island center.

- **delta(...)**: takes 1 argument; returns 1.0 if the argument is the integer 0. It's not an entirely necessary function, the lines of code where it is used could be replaced by something which does not make use of this function (which I might do).
- **Bfield(...)**: takes 3 arguments, two specify the position at which the magnetic field is to be evaluated (one is a 2-vector), and the other one contains the structure of type **fieldparams** containing all the information necessary to evaluate the magnetic field; returns a structure of type **field** containing the magnetic field, its first and second derivatives in the poloidal plane specified by the toroidal angle. It calls the functions **BReim(...)** and **Bcoil(...)** if the magnetic field type is **Reim** or **coil** respectively. The evaluation of the magnetic field for type **heli** is carried out in this function directly, although this can (and I originally intended to) make this into a separate function as well to be called here.
- **gradBReim(...)**: takes 9 arguments, three of which specify the position at which the magnetic field is evaluated and the rest specify the parameters of the magnetic field of type **Reim**; returns a **pointer** to $x$ structures of type **field** containing the gradients, with respect to the $x$ parameters of the magnetic field, of the magnetic field and its first derivative in the poloidal plane.
- **shapeBcoil(...)**: takes 3 arguments, 2 specify the position at which the magnetic field is evaluated (one is a 2-vector) and the remaining one specifies the position coordinates of the coil segment with respect to which the shape gradient of the magnetic field is evaluated; returns a **pointer** to three structures of type **field** containing the shape gradient of the magnetic field at the given position with respect to the three coordinates of the given coil segment.
- **gradBheli(...)**: takes 5 arguments, two specify the position at which the magnetic field is to be evaluated (one is a 2-vector), and the remaining 3 contains the magnetic field parameters; returns a **pointer** to $x$ structures of type **field** containing the gradient of the magnetic field and its first poloidal derivative with respect to the $x$ parameters of the helical coil configuration.
- **gradBfield(...)**: takes 5 arguments, two specify the position at which the magnetic field is to be evaluated (one is a 2-vector), and the other three contain the magnetic field parameters; returns a **pointer** to $x$ structures of type **field** containing the gradient of the magnetic field and its first poloidal derivative with respect to the $x$ parameters of the specified magnetic field type. It calls the functions **gradBReim(...)** and **shapeBcoil(...)** and **gradBheli(...)** if the magnetic field type is **Reim** or **coil** or **heli** respectively.

- **linetodata.c**: **linetodata(...)** takes two arguments, a string containing the line of a file, and a pointer to an integer; returns a **pointer** to an array of **doubles**, obtained from the string.
- **findcentre.c**: contains several functions used to solve for periodic magnetic field lines, tangent maps and adjoint variables:
  - **rho(...)**: takes 3 arguments including the fixed point index; returns an **integer** corresponding to the reordered fixed point index.
  - **inverserho(...)**: takes 3 arguments including the reordered fixed point index; returns an **integer** corresponding to the fixed point index.
  - **printstructposition(...)**: takes 2 arguments including a string and the structure of type position; void function that prints the name of the structure and its elements.

– **printstructfield(. . . )**: takes 2 arguments including a string and the structure of type field; void function that prints the name of the structure and its elements.

– **solve_magneticaxis(. . . )**: takes 5 arguments including field parameters and island center guess; returns the structure type position containing the magnetic axis. *This function has become redundant and may be eliminated.*

– **solve_magneticaxis_save(. . . )**: takes 6 arguments including field parameters and island center guess; **void** function which stores the magnetic axis position and tangent map in a pointer to structures of type position, and the magnetic field and its first and second poloidal derivatives on the magnetic axis in a pointer to structures of type field.

– **extsolve_periodicfieldline(. . . )**: takes 12 arguments including field parameters and a guess for the periodic field line position; returns an **integer** which is 0 if the periodic field line has been found successfully and 1 otherwise, and assigns several values, containing information about the periodic magnetic field line and the associated island chain, to pointers that are passed to this function. The pointer to structures of type **ext_position** contains all the information about the periodic field line (positions, tangent maps, residue, width) at each iterate.

– **solve_lambda_circ(. . . )**: takes 4 arguments including the structure of type **ext_position**; returns a structure of type **position** containing the initial value of the adjoint variable corresponding to the circumference (sum of chords).

– **solve_mu_tangent(. . . )**: takes 5 arguments including the structure of type **ext_position**; returns a pointer to pointers to structures of type **position** containing the values of the adjoint variable $\boldsymbol{\mu}$ of the quantity $\Sigma$ (related to the tangent map) corresponding to the $k$th iterate (out of $L$) and the $Q$th turn around the periodic field line (out of $Q_0$); returns

– **solve_lambda_tangent(. . . )**: takes 7 arguments including the structure of type **ext_position** and the structure of type **position** containing the adjoint variable of the circumference; returns a pointer to pointers to structures of type position containing the values of the adjoint variable $\boldsymbol{\lambda}_{k,Q}$ of the quantity $\Sigma$ (related to the tangent map) corresponding to the $k$th iterate (out of $L$) and the $Q$th turn around the periodic field line (out of $Q_0$).

– **solve_lambda_Res(. . . )**: takes 6 arguments including the structure of type **position** and the structure of type **position** containing the adjoint variable of the circumference; **void** function that fills a pointer to pointers to structures of type position containing the values of the adjoint variable $\boldsymbol{\lambda}_{\mathcal{R}}$ of the residue of the fixed point.

– **solve_lambdaXp(. . . )**: takes 6 arguments

– **solve_gradXp(. . . )**: takes 9 arguments; **void** function that fills a **pointer** to 2 **doubles** corresponding to the gradient of the fixed point position.

– **solve_gradcirc(. . . )**: takes 9 arguments; **void** function that fills a **pointer** to a **double** corresponding to the gradient of the circumference (sum of chords).

– **solve_gradtangent(. . . )**: takes 11 arguments; **void** function that fills a **pointer** to pointers to **doubles** corresponding to the gradient of $\Sigma$ (quantity related to tangent map.

– **solve_gradRes(. . . )**: takes 9 arguments; **void** function that fills a **pointer** to a **double** corresponding to the gradient of the residue $\mathcal{R}$.

- **RungeKutta.c**: contains functions that perform a Runge Kutta step forward in the equations of findcentre.c:
  - **derivative_center(...)**: takes 2 arguments, the magnetic field value and position; returns a **structure** of type **position** containing the right hand side of the equation of motion of the periodic field line position and tangent map.
  - **derivative_center(...)**: takes 3 arguments, the magnetic field value and position, and the displacement vector $\mathbf{s}_\perp$ from the periodic field line; returns a **structure** of type **position** containing the right hand side of the equation of motion of the displacement from the periodic field line position.
  - **derivative_lambdacirc_mutangent(...)**: takes 3 arguments, the magnetic field value and position, and either the adjoint variable $\boldsymbol{\lambda}_C$ or the adjoint variable $\boldsymbol{\mu}_\Sigma$ from the periodic field line; returns a **structure** of type **position** containing the right hand side of the equation of motion of the displacement from the periodic field line position.
  - **derivative_lambdatangent(...)**: takes 5 arguments, the magnetic field value and position, the adjoint variable $\boldsymbol{\lambda}_\Sigma$, the adjoint variable $\boldsymbol{\mu}_\Sigma$ and $\mathbf{s}_\perp$; returns a **structure** of type **position** containing the right hand side of the equation of motion of the adjoint equation of $\boldsymbol{\lambda}_\Sigma$.
  - **derivative_lambdamu(...)**: takes 3 arguments, the magnetic field value and position, the adjoint variable $\boldsymbol{\lambda}_\Sigma$, the adjoint variable $\boldsymbol{\mu}_\Sigma$ and $\mathbf{s}_\perp$; returns a **structure** of type **position** containing the right hand side of the equation of motion of the adjoint equation of $\boldsymbol{\lambda}_\Sigma$.
  - **derivative_gradcirc(...)**: takes 5 arguments, the magnetic field value/derivatives and position/tangent map, the gradient of the magnetic field value/derivative, the adjoint variable $\lambda_C$ and the number $x$ of parameters with respect to which the gradient is taken; returns a **pointer** to $x$ **doubles** containing the right hand side of the equation for the gradient of the circumference with respect to the $x$ parameters.
  - **derivative_gradXp(...)**: takes 5 arguments, the magnetic field value/derivatives and position/tangent map, the gradient of the magnetic field value/derivative, the adjoint variable $\lambda_C$ and the number $x$ of parameters with respect to which the gradient is taken; returns a **pointer** to $x$ **doubles** containing the right hand side of the equation for the gradient of the circumference with respect to the $x$ parameters.
  - **derivative_gradtangent(...)**: takes 7 arguments, the magnetic field value/derivatives and position/tangent map, the gradient of the magnetic field value/derivative, the adjoint variable $\lambda_C$ and the number $x$ of parameters with respect to which the gradient is taken; returns a **pointer** to $x$ **doubles** containing the right hand side of the equation for the gradient of the circumference with respect to the $x$ parameters.
  - **derivative_gradRes(...)**: takes 5 arguments, the magnetic field value/derivatives and position/tangent map, the gradient of the magnetic field value/derivative, the adjoint variable $\lambda_C$ and the number $x$ of parameters with respect to which the gradient is taken; returns a **pointer** to $x$ **doubles** containing the right hand side of the equation for the gradient of the circumference with respect to the $x$ parameters.
  - **addstructs(...)**: takes 4 arguments.
  - **addstructsreassign(...)**: takes 4 arguments.
  - **addstructsfield(...)**: takes 4 arguments.
  - **RK4step(...)**: takes 5 arguments.

  - **RK4stepsave(. . . )**: takes 6 arguments.
  - **RK4step_withsavedB(. . . )**: takes 4 arguments.
  - **RK4step_lambdacirc_mutangent(. . . )**: takes 5 arguments.
  - **RK4step_lambdatangent(. . . )**: takes 7 arguments.
  - **RK4step_lambdaRes(. . . )**: takes 6 arguments.
  - **RK4step_gradcirc(. . . )**: takes 6 arguments.
  - **RK4step_gradRes(. . . )**: takes 9 arguments.
  - **RK4step_gradXp(. . . )**: takes 9 arguments.

- **iota_Poincare.c**: contains **iotaprofile(. . . )** function that calculates the iota profile and the Poincaré plot.
- **linalg2x2.c**: contains functions that perform a Runge Kutta step forward in the equations of findcentre.c:
  - **printmat(. . . )**: takes 4 arguments; **void** function printing a matrix.
  - **linalg2x2(. . . )**: takes 5 arguments; **void** function that calculates the determinant, trace, eigenvalues and eigenvectors.
  - **set_identity(. . . )**: takes no arguments; function returning a **pointer to pointers** to two **doubles** which is the identity matrix.
  - **set_zeros(. . . )**: takes no arguments; function returning a **pointer to pointers** to two **doubles** which is the zero matrix.
  - **invert2x2(. . . )**: takes 2 arguments; function returning a **pointer to pointers** to two **doubles** which is the inverse of the matrix in the argument.
  - **adj2x2(. . . )**: takes 2 argument; function returning a **pointer to pointers** to two **doubles** which is the adjoint of the matrix in the argument.
  - **multiply2x2(. . . )**: takes 3 arguments; returns a **pointer to pointers** to two **doubles** which is the product of two matrices in the arguments.
  - **multiply2x2reassign(. . . )**: takes 3 arguments; **void** function which assigns to one of the arguments a **pointer to pointers** to two **doubles** which is the product of two matrices in the arguments.
  - **multiply(. . . )**: takes 2 arguments; returns a **pointer** to two **doubles** which is the product of the matrix and the vector in the arguments.
  - **symmeigs(. . . )**: takes 4 arguments; **void** function that calculates the symmetrized eigenvectors and eigenvalues of the matrix in the argument.
  - **inner(. . . )**: takes 3 arguments; returns a **double** which is the inner product $\mathbf{v_1} \cdot M \cdot \mathbf{v}_2$ of the 2 vectors $\mathbf{v}_1$ and $\mathbf{v}_2$ and the matrix $M$.
- **magfield_Dommaschk.c**: contains functions that compute the magnetic field from Dommaschk potential. This script is currently redundant, although the equations contained within are correct.

# 3 Step-by-step code