BSM phenomenology the computational way

CATALINA ESPINOZA

IF-UNAM / CONACYT

Taller “Más allá del Modelo Estándar y Astropartículas” | IFUNAM | 15 - marzo - 2023

Summary

Introduction:	Will focus on BSM models of Particle Physics, but only on the part of scanning and construction of likelihood profiles of a given model.
Scanning the parameter space (PS):	Will describe a tool to explore the PS of a model computing a composite likelihood function with respect to the observables of interest.
Parsing the raw data and plotting the likelihood profiles:	Will describe a tool to generate the actual plots that constitute the main results of the numerical analysis.
Tutorial:	This is part of planned tutorials under development that will be available elsewhere. This is a summarized version of the tutorial on likelihood profiles, which will be much more detailed.

Phenomenologing the computational way …

MODEL	OBSERVABLES	SCANNING	PROFILES
How to implement a specific BSM model?	How to compute the observables of interest?	How to explore the parameter space of the model efficiently?	How to analyze the raw data and construct likelihood profiles?
Most cases can be done with e.g., SARAH or FeynRules.	Micromegas exploits the capabilities of CalcHEP to compute e.g., x-sections for DD or ID in addition to Ωh^2. Others: HiggsTools, SmodelS, …	Needs tool to implement algorithm for global optimization e.g., Diver.	Needs parsing tool and plotting engine, e.g., Pippi.
Well documented.	Well documented.	Not sufficiently documented.	Not sufficiently documented.

Phenomenologing the computational way …

Will focus on this part!

MODEL	OBSERVABLES	SCANNING	PROFILES
How to implement a specific BSM model?	How to compute the observables of interest?	How to explore the parameter space of the model efficiently?	How to analyze the raw data and construct likelihood profiles?
Most cases can be done with e.g., SARAH or FeynRules.	Micromegas exploits the capabilities of CalcHEP to compute e.g., x-sections for DD or ID in addition to Ωh^2. Others: HiggsTools, SmodelS, …	Needs tool to implement algorithm for global optimization e.g., Diver.	Needs parsing tool and plotting engine, e.g., Pippi.
Well documented.	Well documented.	Not sufficiently documented.	Not sufficiently documented.

Simple BSM model example

Scalar sector	Two SU(2) Higgs doublets H1 and H2 and one singlet φ
	The singlet mixes with the real parts of the neutral components of the doublets into three CP-even physical scalars h1, h2 and h3
Dark sector	One majorana fermion coupled to the singlet: $y_\Psi \, \overline{\Psi}_R^C \, \Psi_R \, \varphi$
	The mixing in the scalar sector induces couplings of the physical scalars with the DM and therefore we have effective DM-quark couplings and thus DM-nucleon scattering

Observables

We calculate numerically the observables and from them the chi-square functions

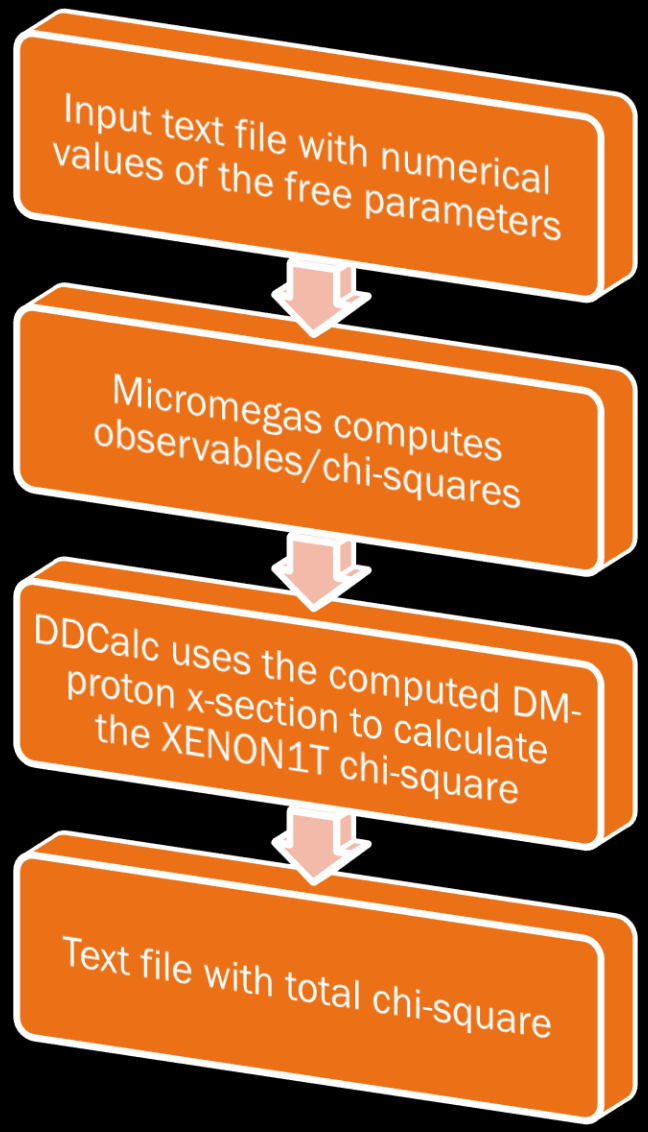
Higgs mass: $\chi_h^2 = (m_h - m_h^{PDG})^2/\sigma_h^2$

Relic density: $\chi_{\Omega h}^2 = (\Omega h^2 - \Omega^{Planck} h^2)^2/\sigma_{\Omega h}^2$

DM-proton x-sect: χ_{DD}^2 = … (from DDCalc tool)

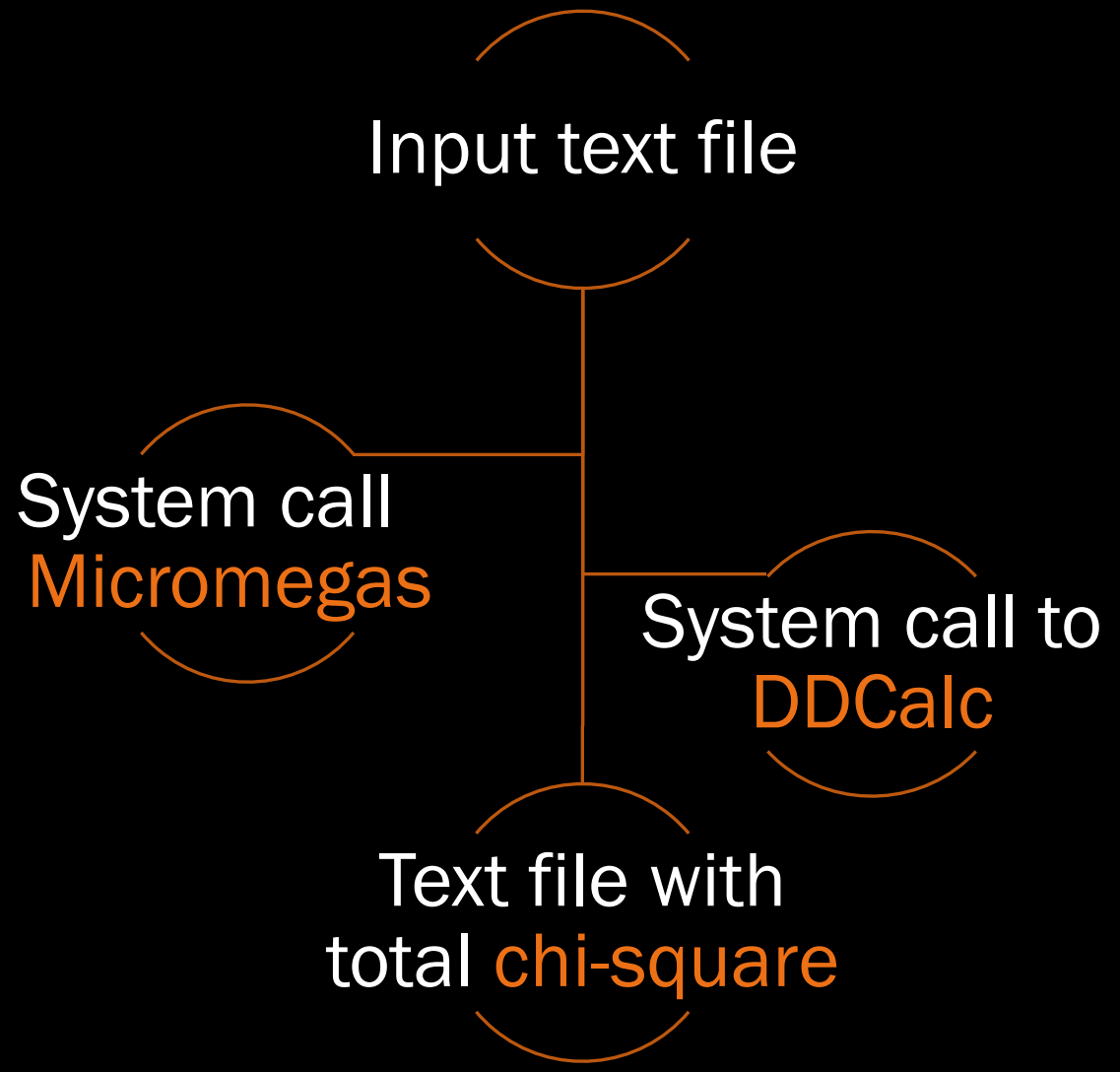
For example…

- Use a text file to pass the numerical values of the free parameters (1 single point of parameter space) to micromegas
- Micromegas computes relic density, higgs mass and DM-proton x-section and writes their values to a text file with also the chi-square from the simple formulas above
- DDCalc reads form the text file the DM-proton x-section and computes the chi-square relative to the XENON-1T experiment, writes the result to a text file

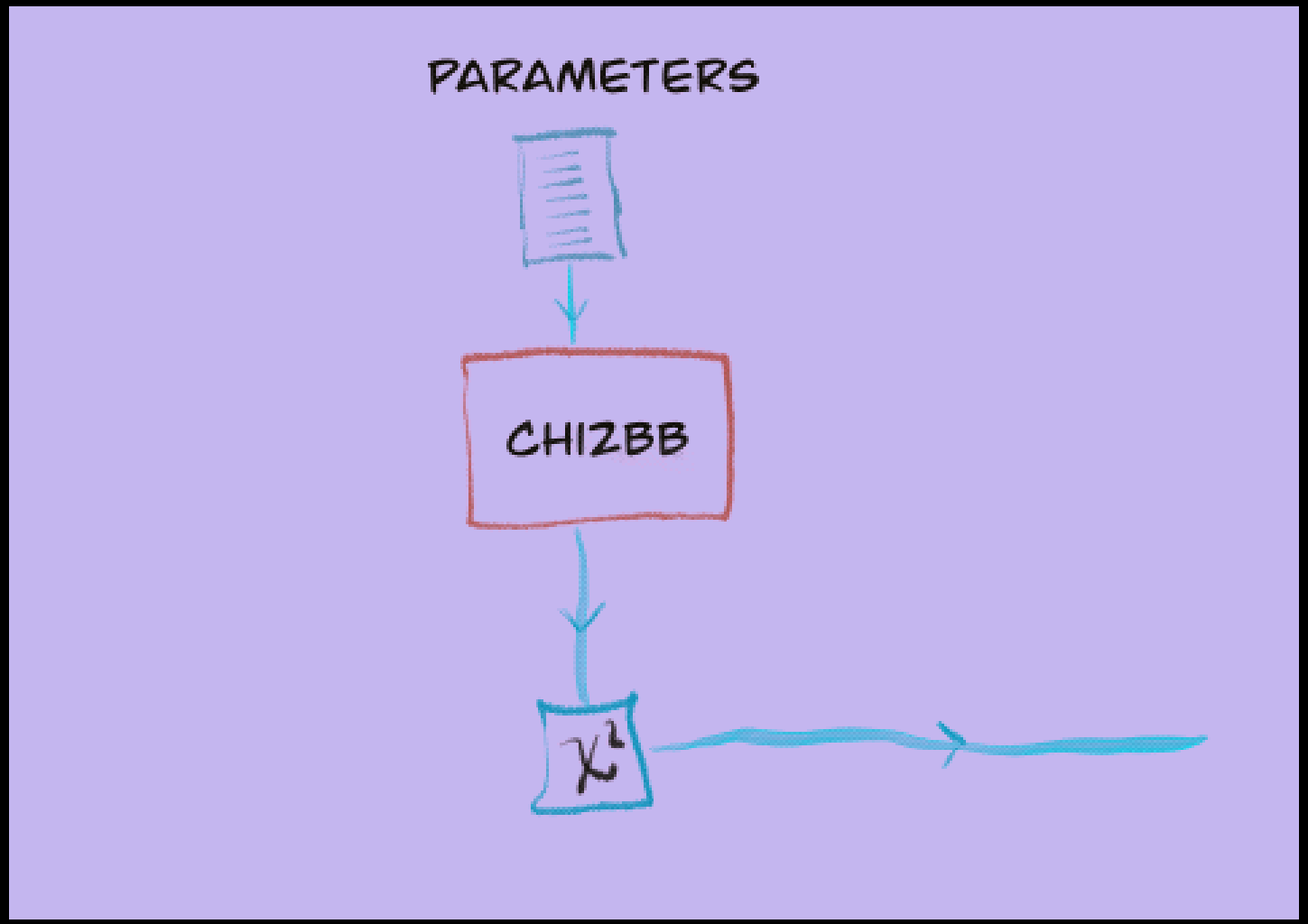


In practice …

- From the scanning program we communicate with Micromegas and DDCalc through “system calls”, for example in C:
- system(“./my_micromegas_prog par_file.txt”)
- This executes Micromegas with input file ‘par_file.txt’ previously created with the values of the free variables for 1 point of parameter space



We call this: Chi-square Black Box



Exploring the parameter space

- Next, we use a global optimization algorithm to explore the parameter space and find the minimum of the composite (total) chi-square function
- Pictorially … suppose we want to explore a 2-dimensional surface

Number of points (NP)

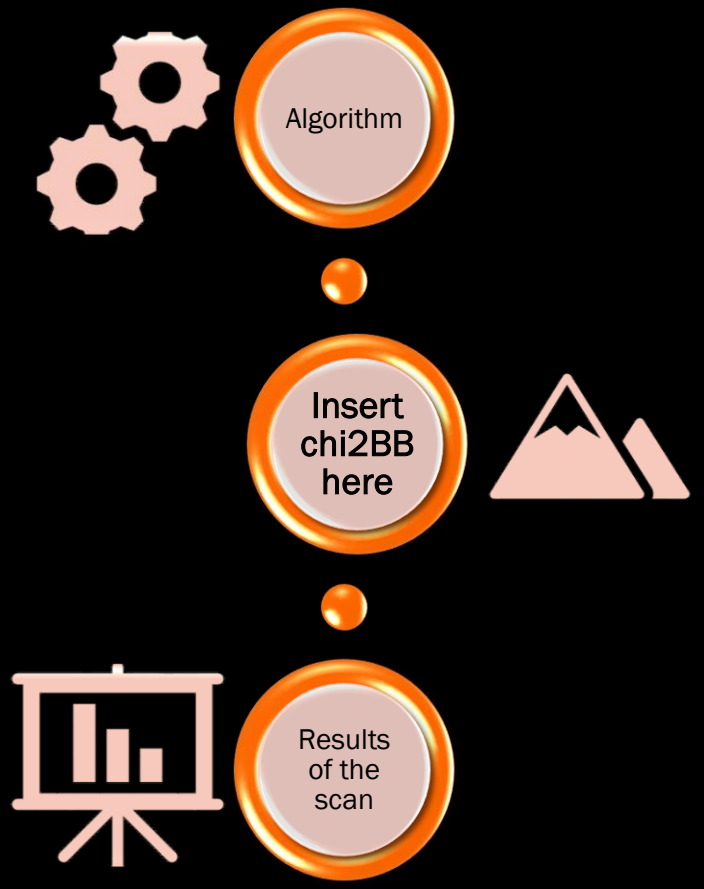
- Choose NP points on the surface randomly
- Using the chi2BB, compute the heights for all of them
- Save to disk the coordinates, observables, heights of all points
- This is called a ‘generation’, with a population of NP points

Clever algorithm

- Use some algorithm to ‘move around’ the points to new locations (a new generation) and repeat
- Stop repeating when certain ‘convergence criteria’ is met (there is a high probability that the global minimum has been found)
- Clearly, the larger the ‘population’ is, the analysis is better
- Typically, NP=20,000 is recommended for dimension ~ 20 or less

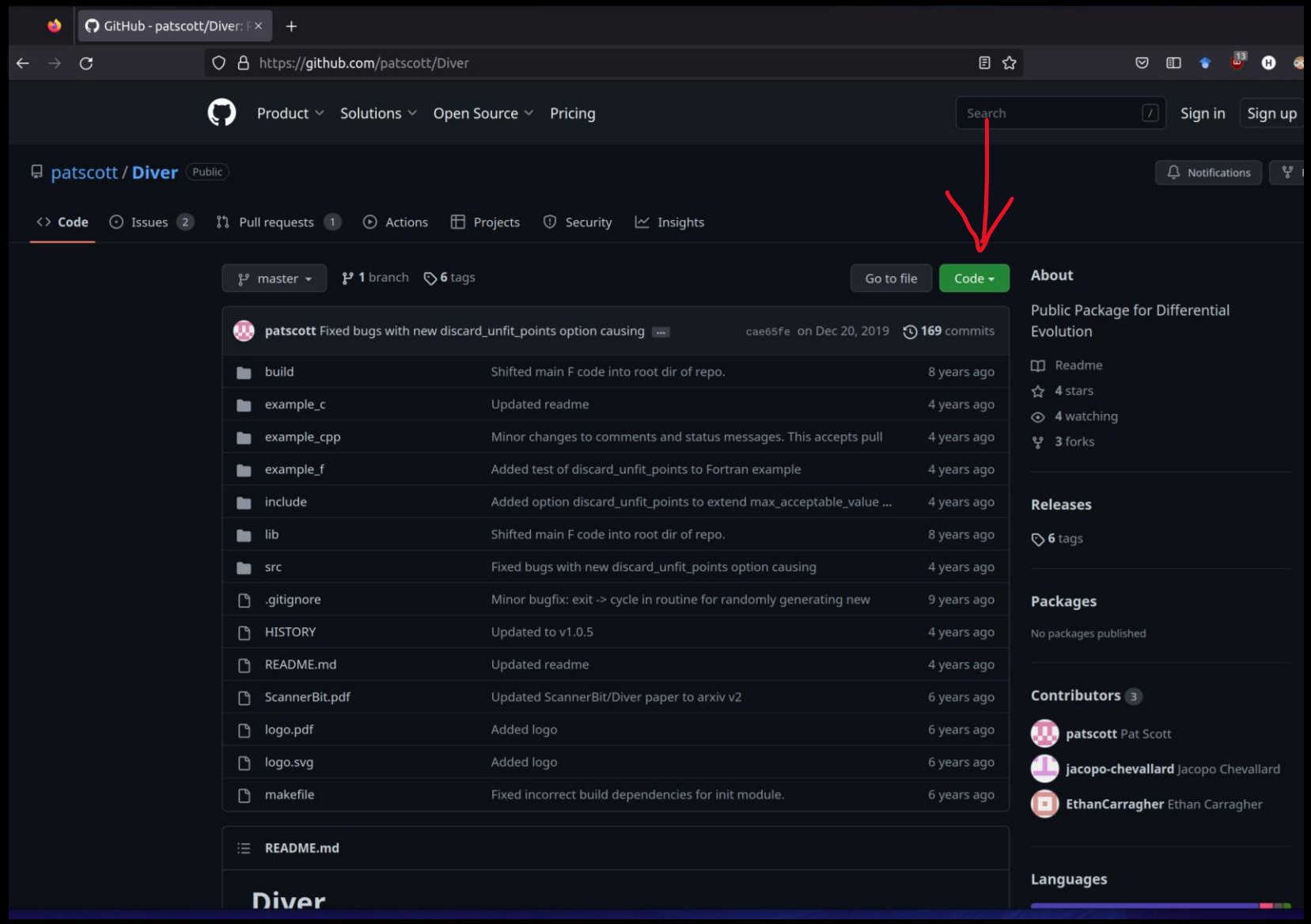
The Global Optimisation Algorithm Black Box

- For practical purposes we don't need to know how the algorithm works: it is just a minimization black box



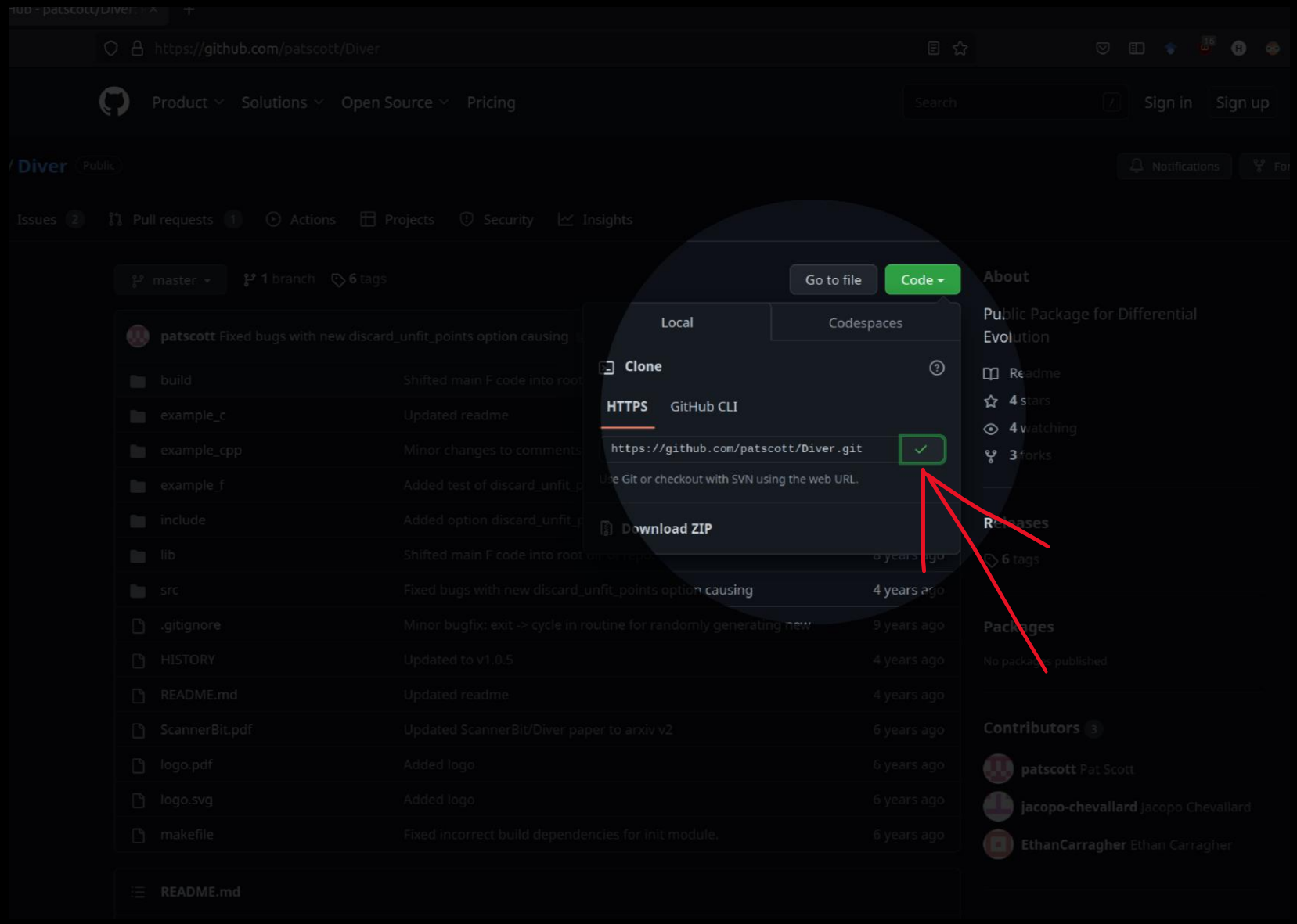
The ‘Diver’ package

- We use this package authored by Pat Scott, which can be obtained from GitHub …



The ‘Diver’ package

- Click on ‘Code’ and then copy the address …



The ‘Diver’ package

- From the ‘Readme’ we see that it compiles with a simple instruction …

Diver is written in Fortran2003. We originally wrote it with applications ir
mind, but there is nothing to prevent its use in other fields.

The code and its options are described in detail in the ScannerBit paper
papers that use results or insights obtained with Diver should cite this p

1. Martinez, McKay, Farmer, Scott, Roebber, Putze & Conrad 2017, Euro
   arXiv:1705.07959

Compilation

The Diver build system is not really complex enough to require autotool
by hand to suit your system, or call it from another makefile or the comr

To build Diver as a static library, and build all examples, do

```
make
```

To instead build Diver as a shared library, do

```
make libdiver.so
```

To build only the static library, do

Installing Diver

- This is the complete set of instructions to install Diver
- You may need to install also openmpi in case you don't have it, this is to execute the program using several cores in parallel

```
sudo apt-get -y install openmpi-bin
git clone https://github.com/patscott/Diver.git
cd Diver
make
cd Diver/example_c/
gedit example_c.c &
```

Examples

- The tool has an example of use, the same example is presented written in C, C++ and Fortran
- We open the C source code of the example here …

```
sudo apt-get -y install openmpi-bin
git clone https://github.com/patscott/Diver.git
cd Diver
make
cd Diver/example_c/
gedit example_c.c &
```

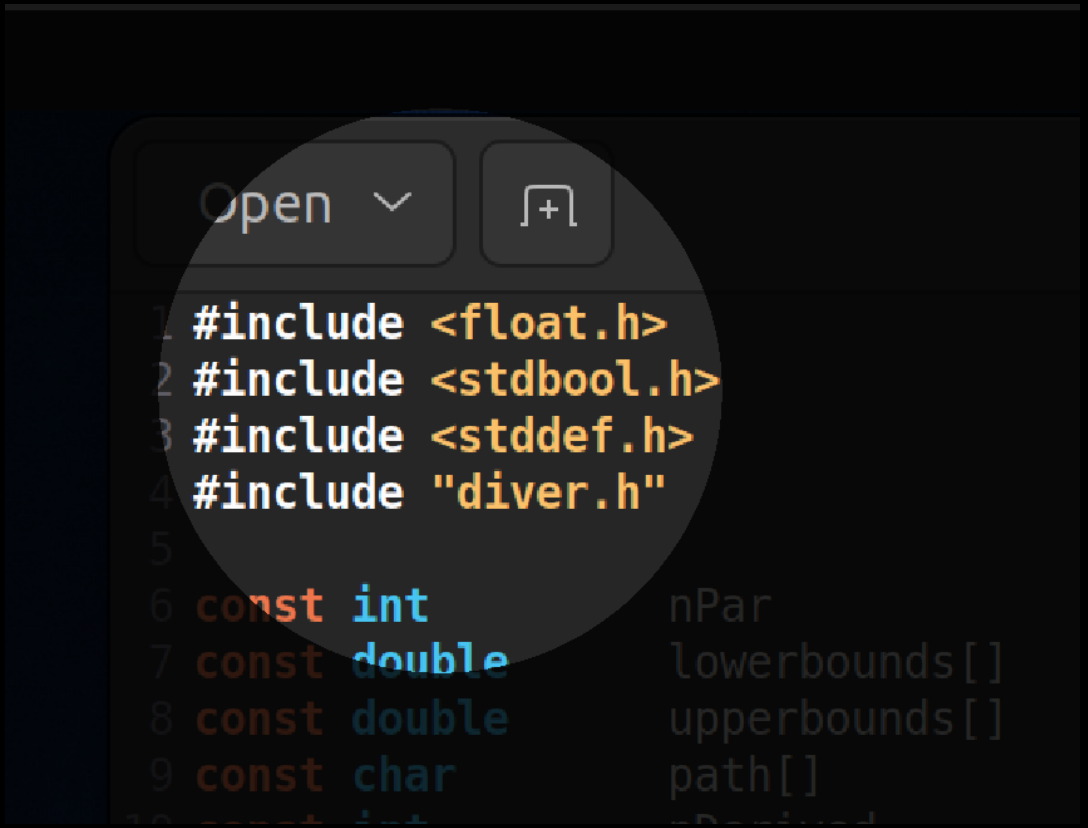
The C example

- This is the entire program
- The idea is to adapt this code to scan our model
- We can divide the code's parts in four blocks ...

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int         nPar                = 5;                          // Dimensionality of the parameter space
const double      lowerbounds[]       = {-5.,-50.,-5.,-50.,-2.};    // Lower boundaries of parameter space
const double      upperbounds[]       = { 5., 50., 5., 50., 2.};    // Upper boundaries of parameter space
const char        path[]              = "example_c/output/example"; // Path to save samples, resume files, etc
const int         nDerived            = 0;                          // Number of derived quantities to output
const int         nDiscrete           = 0;                          // Number of parameters that are to be treated as discrete
const int         discrete[]          = {};                         // Indices of discrete parameters, Fortran style, i.e. starting
const bool        partitionDiscrete   = false;                      // Split the population evenly amongst discrete parameters and
const int         maxciv              = 1;                          // Maximum number of civilisations
const int         maxgen              = 100;                        // Maximum number of generations per civilisation
const int         NP                  = 1000;                       // Population size (individuals per generation)
const int         nF                  = 1;                          // Size of the array indicating scale factors
const double      F[]                 = {0.6};                      // Scale factor(s).  Note that this must be entered as an array
const double      Cr                  = 0.9;                        // Crossover factor
const double      lambda              = 0.8;                        // Mixing factor between best and rand/current
const bool        current             = false;                      // Use current vector for mutation
const bool        expon               = false;                      // Use exponential crossover
const int         bndry               = 3;                          // Boundary constraint: 1=brick wall, 2=random re-initializatio
const bool        jDE                 = true;                       // Use self-adaptive choices for rand/1/bin parameters as per B
const bool        lambdajDE           = true;                       // Use self-adaptive rand-to-best/1/bin parameters; based on Br
const double      convthresh          = 1.e-6;                      // Threshold for gen-level convergence: smoothed fractional imp
const int         convsteps           = 10;                         // Number of steps to smooth over when checking convergence
const bool        removeDuplicates    = true;                       // Weed out duplicate vectors within a single generation
const bool        doBayesian          = false;                      // Calculate approximate log evidence and posterior weightings
const double      maxNodePop          = 1.9;                        // Population at which node is partitioned in binary space part
const double      Ztolerance          = 1.e-3;                      // Input tolerance in log-evidence
const int         savecount           = 100;                        // Save progress every savecount generations
const bool        resume              = false;                      // Restart from a previous run
const bool        outputSamples       = false;                      // Write output .raw and .sam (if nDerived != 0) files
const int         init_pop_strategy   = 0;                          // Initialisation strategy: 0=one shot, 1=n-shot, 2=n-shot with
const bool        discard_unfit_points = false;                     // Recalculate any trial vector whose fitness is above max_acce
const int         max_init_attempts   = 10000;                      // Maximum number of times to try to find a valid vector for ea
const double      max_acceptable_val  = 1e6;                        // Maximum fitness to accept for the initial generation if init
const int         seed                = 1234567;                    // base seed for random number generation; non-positive or abse
const int         verbose             = 1;                          // Output verbosity: 0=only error messages, 1=basic info, 2=civ


//Function to be minimized.  Corresponds to -ln(Likelihood).
//Plain Gaussian centred at the origin. Valid for any number of dimensions.  Minimum value is the number of dimensions.
double gauss(double params[], const int param_dim, int *fcall, bool *quit, const bool validvector, void** context)
{
  double result = 0.0;
  for (int i = 0; i<param_dim; i++) result += params[i]*params[i] + 1.0;
  if (!validvector) result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
  void* context = &gauss; //Not actually used in this example.
  cdiver(gauss, nPar, lowerbounds, upperbounds, path, nDerived, nDiscrete, discrete, partitionDiscrete,
         maxciv, maxgen, NP, nF, F, Cr, lambda, current, expon, bndry, jDE, lambdajDE, convthresh,
         convsteps, removeDuplicates, doBayesian, NULL, maxNodePop, Ztolerance, savecount, resume,
         outputSamples, init_pop_strategy, discard_unfit_points, max_init_attempts, max_acceptable_val, seed, context, verbose);
         //Note that prior, maxNodePop and Ztolerance are just ignored if doBayesian = false
}
```

The first block

- The first part is the ‘include’ block
- Here you add libraries that you need, e.g. ‘stdio.h’ for handling files



The fourth block

- The last part is the ‘main’ function
- This part does not need to be modified
- Just make sure that the name of the chi2BB function (here ‘gauss’) is consistent with the third block as describe next …

```
  for (int i = 0; i<param_dim; i++) resu
  if (!validvector) result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
  void* context = &gauss; //Not actually
  cdiver(gauss, nPar, lowerbounds, upper
         maxciv, maxgen, NP, nF, F, Cr,
         convsteps, removeDuplicates, do
         outputSamples, init_pop_strateg
         //Note that prior, maxNodePop a
}
```

The third block

- The third block is the function that is being minimized
- We need to replace this with our chi2BB code in order to scan the parameter space of our model

```
const int          seed
const int          verbose


//Function to be minimized.  Corresp
//Plain Gaussian centred at the orig
double gauss(double params[], const
{
  double result = 0.0;
  for (int i = 0; i<param_dim; i++)
  if (!validvector) result = DBL_MAX
  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
```

The third block

- We need to comment one line first …

```
39 const int          seed
40 const int          verbose
41
42
43 //Function to be minimized.  Corresp
44 //Plain Gaussian centred at the orig
45 double gauss(double params[], const
46 {
47   double result = 0.0;
48   for (int i = 0; i<param_dim; i++)
49   if (!validvector) result = DBL_MA
50   *fcall += 1;
51   *quit = false;
52   return result;
53 }
54
55
56 int main(int argc, char** argv)
57 {
```

Before

```
39 const int          seed                    = 1234567;
40 const int          verbose                 = 1;
41
42
43 //Function to be minimized.  Corresponds to -ln(Like
44 //Plain Gaussian centred at the origin. Valid for a
45 double gauss(double params[], const int param_dim, i
46 {
47   double result = 0.0;
48   //for (int i = 0; i<param_dim; i++) result += para
49   if (!validvector)
50   {
51   result = DBL_MAX;
52   *fcall += 1;
53   *quit = false;
54   }
55
56   // Insert here code for chi2BB
57   //
58   // ......
59   //
60   //
61
62   *fcall += 1;
63   *quit = false;
64   return result;
65 }
66
67
68 int main(int argc, char** argv)
69 {
```

After

The third block

- Then we group the next 3 lines in a single ‘if’ statement …

```
39 const int          seed
40 const int          verbose
41
42
43 //Function to be minimized.  Corresp
44 //Plain Gaussian centred at the orig
45 double gauss(double params[], const
46 {
47   double result = 0.0;
48   for (int i = 0; i<param_dim; i++)
49   if (!validvector) result = DBL_MAX
50   *fcall += 1;
51   *quit = false;
52   return result;
53 }
54
55
56 int main(int argc, char** argv)
57 {
```

Before

```
39 const int          seed                    = 1234567;
40 const int          verbose                 = 1;
41
42
43 //Function to be minimized.  Corresponds to -ln(Like
44 //Plain Gaussian centred at the origin. Valid for ar
45 double gauss(double params[], const int param_dim, i
46 {
47   double result = 0.0;
48   //for (int i = 0; i<param_dim; i++) result += para
49   if (!validvector)
50   {
51   result = DBL_MAX;
52   *fcall += 1;
53   *quit = false;
54   }
55
56   // Insert here code for chi2BB
57   //
58   // ......
59   //
60   //
61
62   *fcall += 1;
63   *quit = false;
64   return result;
65 }
66
67
68 int main(int argc, char** argv)
69 {
```

After

The third block

- Then we copy the 2 shown lines just before the ‘return’ statement …

```
const int          seed
const int          verbose


//Function to be minimized.  Corresp
//Plain Gaussian centred at the orig
double gauss(double params[], const
{
  double result = 0.0;
  for (int i = 0; i<param_dim; i++)
  if (!validvector) result = DBL_MA
  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
```

Before

```
const int          seed                    = 1234567;
const int          verbose                 = 1;


//Function to be minimized.  Corresponds to -ln(Like
//Plain Gaussian centred at the origin. Valid for a
double gauss(double params[], const int param_dim, i
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result += para
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
```

After

The third block

- Finally, we write our code for the chi2BB in the space shown …

```
const int          seed
const int          verbose


//Function to be minimized.  Corresp
//Plain Gaussian centred at the orig
double gauss(double params[], const
{
  double result = 0.0;
  for (int i = 0; i<param_dim; i++)
  if (!validvector) result = DBL_MAX
  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
```

Before

```
const int          seed                    = 1234567;
const int          verbose                 = 1;


//Function to be minimized.  Corresponds to -ln(Like
//Plain Gaussian centred at the origin. Valid for ar
double gauss(double params[], const int param_dim, i
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result += para
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
{
```

After

The second block

- But first let me describe the second block
- In this block, global variables are defined in order to configure Diver
- In most cases, almost all variables can be taken with the default values
- I will describe those that need to be changed according to our model …
- Only the values in the ‘pink’ column must be changed …

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int      nPar                 = 5;                          // Dimensio
const double   lowerbounds[]        = {-5.,-50.,-5.,-50.,-2.};    // Lower bo
const double   upperbounds[]        = { 5., 50., 5., 50., 2.};    // Upper bo
const char     path[]               = "example_c/output/example"; // Path to
const int      nDerived             = 0;                          // Number o
const int      nDiscrete            = 0;                          // Number o
const int      discrete[]           = {};                         // Indices
const bool     partitionDiscrete    = false;                      // Split th
const int      maxciv               = 1;                          // Maximum
const int      maxgen               = 100;                        // Maximum
const int      NP                   = 1000;                       // Populati
const int      nF                   = 1;                          // Size of
const double   F[]                  = {0.6};                      // Scale fa
const double   Cr                   = 0.9;                        // Crossove
const double   lambda               = 0.8;                        // Mixing f
const bool     current              = false;                      // Use curr
const bool     expon                = false;                      // Use expo
const int      bndry                = 3;                          // Boundary
const bool     jDE                  = true;                       // Use self
const bool     lambdajDE            = true;                       // Use self
const double   convthresh           = 1.e-6;                      // Threshol
const int      convsteps            = 10;                         // Number o
const bool     removeDuplicates     = true;                       // Weed out
const bool     doBayesian           = false;                      // Calculat
const double   maxNodePop           = 1.9;                        // Populati
const double   Ztolerance           = 1.e-3;                      // Input to
const int      savecount            = 100;                        // Save pro
const bool     resume               = false;                      // Restart
const bool     outputSamples        = false;                      // Write ou
const int      init_pop_strategy    = 0;                          // Initiali
const bool     discard_unfit_points = false;                      // Recalcul
const int      max_init_attempts    = 10000;                      // Maximum
const double   max_acceptable_val   = 1e6;                        // Maximum
const int      seed                 = 1234567;                    // base see
const int      verbose              = 1;                          // Output v

//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- nPar needs to be adjusted according to the number of free parameters in the model
- The next 2 arrays have the values of the intervals in which your free parameters can take values, you choose the order of your parameters
- In this example, the first parameter can be varied between -5 and 5, the second between -50 and 50, etc.

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int       nPar                = 5;
const double    lowerbounds[]       = {-5.,-50.,-5.,-50.,-2.};
const double    upperbounds[]       = { 5., 50., 5., 50., 2.};
const char      path[]              = "example_c/output/example";
const int       nDerived            = 0;
const int       nDiscrete           = 0;
const int       discrete[]          = {};
const bool      partitionDiscrete   = false;
const int       maxciv              = 1;
const int       maxgen              = 100;
const int       NP                  = 1000;
const int       nF                  = 1;
const double    F[]                 = {0.6};
const double    Cr                  = 0.9;
const double    lambda              = 0.8;
const bool      current             = false;
const bool      expon               = false;
const int       bndry               = 3;
const bool      jDE                 = true;
const bool      lambdajDE           = true;
const double    convthresh          = 1.e-6;
const int       convsteps           = 10;
const bool      removeDuplicates    = true;
const bool      doBayesian          = false;
const double    maxNodePop          = 1.9;
const double    Ztolerance          = 1.e-3;
const int       savecount           = 100;
const bool      resume              = false;
const bool      outputSamples       = false;
const int       init_pop_strategy   = 0;
const bool      discard_unfit_points = false;
const int       max_init_attempts   = 10000;
const double    max_acceptable_val  = 1e6;
const int       seed                = 1234567;
const int       verbose             = 1;

//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- Next, the directory where the data will be saved is stated under quotes
- In this example the directory is (relative to the directory where the executable is) example_c/output/
- The files created with the data will have the same name but different extensions, in this example the name is ‘example’

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int      nPar                 = 5;                          // Dimensio
const double   lowerbounds[]        = {-5.,-50.,-5.,-50.,-2.};    // Lower bo
const double   upperbounds[]        = { 5., 50., 5., 50., 2.};    // Upper bo
const char     path[]               = "example_c/output/example"; // Path to
const int      nDerived             = 0;                          // Number o
const int      nDiscrete            = 0;                          // Number o
const int      discrete[]           = {};                         // Indices
const bool     partitionDiscrete    = false;                      // Split th
const int      maxciv               = 1;                          // Maximum
const int      maxgen               = 100;                        // Maximum
const int      NP                   = 1000;                       // Populati
const int      nF                   = 1;                          // Size of
const double   F[]                  = {0.6};                      // Scale fa
const double   Cr                   = 0.9;                        // Crossove
const double   lambda               = 0.8;                        // Mixing f
const bool     current              = false;                      // Use curr
const bool     expon                = false;                      // Use expo
const int      bndry                = 3;                          // Boundary
const bool     jDE                  = true;                       // Use self
const bool     lambdajDE            = true;                       // Use self
const double   convthresh           = 1.e-6;                      // Threshol
const int      convsteps            = 10;                         // Number o
const bool     removeDuplicates     = true;                       // Weed out
const bool     doBayesian           = false;                      // Calculat
const double   maxNodePop           = 1.9;                        // Populati
const double   Ztolerance           = 1.e-3;                      // Input to
const int      savecount            = 100;                        // Save pro
const bool     resume               = false;                      // Restart
const bool     outputSamples        = false;                      // Write ou
const int      init_pop_strategy    = 0;                          // Initiali
const bool     discard_unfit_points = false;                      // Recalcul
const int      max_init_attempts    = 10000;                      // Maximum
const double   max_acceptable_val   = 1e6;                        // Maximum
const int      seed                 = 1234567;                    // base see
const int      verbose              = 1;                          // Output v


//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- Next, the variable nDerived refers to what we call observables, it is the number of observables that we want to save for each point
- In our case, these are the Higgs mass, the relic density, the DM mass, the DM-proton x-section, 3 chi-square functions and their sum
- Therefore, we should put nDerived = 8;

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int        nPar                = 5;                                  // Dimensio
const double     lowerbounds[]       = {-5.,-50.,-5.,-50.,-2.};            // Lower bo
const double     upperbounds[]       = { 5., 50., 5., 50., 2.};            // Upper bo
const char       path[]              = "example_c/output/example";         // Path to
const int        nDerived            = 0;                                  // Number o
const int        nDiscrete           = 0;                                  // Number o
const int        discrete[]          = {};                                 // Indices
const bool       partitionDiscrete   = false;                              // Split th
const int        maxciv              = 1;                                  // Maximum
const int        maxgen              = 100;                                // Maximum
const int        NP                  = 1000;                               // Populati
const int        nF                  = 1;                                  // Size of
const double     F[]                 = {0.6};                              // Scale fa
const double     Cr                  = 0.9;                                // Crossove
const double     lambda              = 0.8;                                // Mixing f
const bool       current             = false;                              // Use curr
const bool       expon               = false;                              // Use expo
const int        bndry               = 3;                                  // Boundary
const bool       jDE                 = true;                               // Use self
const bool       lambdajDE           = true;                               // Use self
const double     convthresh          = 1.e-6;                              // Threshol
const int        convsteps           = 10;                                 // Number o
const bool       removeDuplicates    = true;                               // Weed out
const bool       doBayesian          = false;                              // Calculat
const double     maxNodePop          = 1.9;                                // Populati
const double     Ztolerance          = 1.e-3;                              // Input to
const int        savecount           = 100;                                // Save pro
const bool       resume              = false;                              // Restart
const bool       outputSamples       = false;                              // Write ou
const int        init_pop_strategy   = 0;                                  // Initiali
const bool       discard_unfit_points = false;                             // Recalcul
const int        max_init_attempts   = 10000;                              // Maximum
const double     max_acceptable_val  = 1e6;                                // Maximum
const int        seed                = 1234567;                            // base see
const int        verbose             = 1;                                  // Output v

//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- Next, the variable maxgen should be put equal to a large number to ensure that a sufficiently large number of ‘generations’ are generated
- If maxgen is small the calculation might be stopped before the convergence criteria is met
- We recommend something like maxgen = 10000;
- NP is the population; the recommended value is 20,000

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int        nPar                = 5;
const double     lowerbounds[]       = {-5.,-50.,-5.,-50.,-2.};
const double     upperbounds[]       = { 5., 50., 5., 50., 2.};
const char       path[]              = "example_c/output/example";
const int        nDerived            = 0;
const int        nDiscrete           = 0;
const int        discrete[]          = {};
const bool       partitionDiscrete   = false;
const int        maxciv              = 1;
const int        maxgen              = 100;
const int        NP                  = 1000;
const int        nF                  = 1;
const double     F[]                 = {0.6};
const double     Cr                  = 0.9;
const double     lambda              = 0.8;
const bool       current             = false;
const bool       expon               = false;
const int        bndry               = 3;
const bool       jDE                 = true;
const bool       lambdajDE           = true;
const double     convthresh          = 1.e-6;
const int        convsteps           = 10;
const bool       removeDuplicates    = true;
const bool       doBayesian          = false;
const double     maxNodePop          = 1.9;
const double     Ztolerance          = 1.e-3;
const int        savecount           = 100;
const bool       resume              = false;
const bool       outputSamples       = false;
const int        init_pop_strategy   = 0;
const bool       discard_unfit_points = false;
const int        max_init_attempts   = 10000;
const double     max_acceptable_val  = 1e6;
const int        seed                = 1234567;
const int        verbose             = 1;


//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- Next, the variable ‘convthresh’ is used for the convergence criteria, it is recommended to be set at the value 1e-4
- Smaller values might take too much computing time, while greater values might lead to incomplete analysis of the parameter space

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int      nPar                 = 5;                                  // Dimensio
const double   lowerbounds[]        = {-5.,-50.,-5.,-50.,-2.};            // Lower bo
const double   upperbounds[]        = { 5., 50., 5., 50., 2.};            // Upper bo
const char     path[]               = "example_c/output/example";         // Path to
const int      nDerived             = 0;                                  // Number o
const int      nDiscrete            = 0;                                  // Number o
const int      discrete[]           = {};                                 // Indices
const bool     partitionDiscrete    = false;                              // Split th
const int      maxciv               = 1;                                  // Maximum
const int      maxgen               = 100;                                // Maximum
const int      NP                   = 1000;                               // Populati
const int      nF                   = 1;                                  // Size of
const double   F[]                  = {0.6};                              // Scale fa
const double   Cr                   = 0.9;                                // Crossove
const double   lambda               = 0.8;                                // Mixing f
const bool     current              = false;                              // Use curr
const bool     expon                = false;                              // Use expo
const int      bndry                = 3;                                  // Boundary
const bool     jDE                  = true;                               // Use self
const bool     lambdajDE            = true;                               // Use self
const double   convthresh           = 1.e-6;                              // Threshol
const int      convsteps            = 10;                                 // Number o
const bool     removeDuplicates     = true;                               // Weed out
const bool     doBayesian           = false;                              // Calculat
const double   maxNodePop           = 1.9;                                // Populati
const double   Ztolerance           = 1.e-3;                              // Input to
const int      savecount            = 100;                                // Save pro
const bool     resume               = false;                              // Restart
const bool     outputSamples        = false;                              // Write ou
const int      init_pop_strategy    = 0;                                  // Initiali
const bool     discard_unfit_points = false;                              // Recalcul
const int      max_init_attempts    = 10000;                              // Maximum
const double   max_acceptable_val   = 1e6;                                // Maximum
const int      seed                 = 1234567;                            // base see
const int      verbose              = 1;                                  // Output v


//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- The variable savecount should always be equal to 1
- This ensures that the observables are saved for all the points explored during the scan
- This is important to generate the contours of equal likelihood

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int      nPar                 = 5;                                  // Dimensio
const double   lowerbounds[]        = {-5.,-50.,-5.,-50.,-2.};            // Lower bo
const double   upperbounds[]        = { 5., 50., 5., 50., 2.};            // Upper bo
const char     path[]               = "example_c/output/example";         // Path to
const int      nDerived             = 0;                                  // Number o
const int      nDiscrete            = 0;                                  // Number o
const int      discrete[]           = {};                                 // Indices
const bool     partitionDiscrete    = false;                              // Split th
const int      maxciv               = 1;                                  // Maximum
const int      maxgen               = 100;                                // Maximum
const int      NP                   = 1000;                               // Populati
const int      nF                   = 1;                                  // Size of
const double   F[]                  = {0.6};                              // Scale fa
const double   Cr                   = 0.9;                                // Crossove
const double   lambda               = 0.8;                                // Mixing f
const bool     current              = false;                              // Use curr
const bool     expon                = false;                              // Use expo
const int      bndry                = 3;                                  // Boundary
const bool     jDE                  = true;                               // Use self
const bool     lambdajDE            = true;                               // Use self
const double   convthresh           = 1.e-6;                              // Threshol
const int      convsteps            = 10;                                 // Number o
const bool     removeDuplicates     = true;                               // Weed out
const bool     doBayesian           = false;                              // Calculat
const double   maxNodePop           = 1.9;                                // Populati
const double   Ztolerance           = 1.e-3;                              // Input to
const int      savecount            = 100;                                // Save pro
const bool     resume               = false;                              // Restart
const bool     outputSamples        = false;                              // Write ou
const int      init_pop_strategy    = 0;                                  // Initiali
const bool     discard_unfit_points = false;                              // Recalcul
const int      max_init_attempts    = 10000;                              // Maximum
const double   max_acceptable_val   = 1e6;                                // Maximum
const int      seed                 = 1234567;                            // base see
const int      verbose              = 1;                                  // Output v

//Function to be minimized.  Corresponds to -ln(Likelihood).
```

The second block

- Finally, the variable ‘outputSamples’ should be equal to ‘true’
- This ensures that the observables are saved to disk alongside the parameters for each point of parameter space explored

```c
#include <float.h>
#include <stdbool.h>
#include <stddef.h>
#include "diver.h"

const int      nPar                 = 5;                                  // Dimensio
const double   lowerbounds[]        = {-5.,-50.,-5.,-50.,-2.};            // Lower bo
const double   upperbounds[]        = { 5., 50., 5., 50., 2.};            // Upper bo
const char     path[]               = "example_c/output/example";         // Path to
const int      nDerived             = 0;                                  // Number o
const int      nDiscrete            = 0;                                  // Number o
const int      discrete[]           = {};                                 // Indices
const bool     partitionDiscrete    = false;                              // Split th
const int      maxciv               = 1;                                  // Maximum
const int      maxgen               = 100;                                // Maximum
const int      NP                   = 1000;                               // Populati
const int      nF                   = 1;                                  // Size of
const double   F[]                  = {0.6};                              // Scale fa
const double   Cr                   = 0.9;                                // Crossove
const double   lambda               = 0.8;                                // Mixing f
const bool     current              = false;                              // Use curr
const bool     expon                = false;                              // Use expo
const int      bndry                = 3;                                  // Boundary
const bool     jDE                  = true;                               // Use self
const bool     lambdajDE            = true;                               // Use self
const double   convthresh           = 1.e-6;                              // Threshol
const int      convsteps            = 10;                                 // Number o
const bool     removeDuplicates     = true;                               // Weed out
const bool     doBayesian           = false;                              // Calculat
const double   maxNodePop           = 1.9;                                // Populati
const double   Ztolerance           = 1.e-3;                              // Input to
const int      savecount            = 100;                                // Save pro
const bool     resume               = false;                              // Restart
const bool     outputSamples        = false;                              // Write ou
const int      init_pop_strategy    = 0;                                  // Initiali
const bool     discard_unfit_points = false;                              // Recalcul
const int      max_init_attempts    = 10000;                              // Maximum
const double   max_acceptable_val   = 1e6;                                // Maximum
const int      seed                 = 1234567;                            // base see
const int      verbose              = 1;                                  // Output v


//Function to be minimized.  Corresponds to -ln(Likelihood).
```

Inserting the chi2BB code

- Let us return to the third block
- We will insert the code for the generation of the composite chi-square for a given point of parameter space

```
const int          verbose              = 1;


//Function to be minimized.  Corresponds to -ln(
//Plain Gaussian centred at the origin. Valid fo
double gauss(double params[], const int param_di
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result +=
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  *fcall += 1;
  *quit = false;
  return result;
}


int main(int argc, char** argv)
```

Inserting the chi2BB code

- This is the resulting code, except that we have not put explicitly the code for the generation of the text files nor the system calls

```
//Function to be minimized.  Corresponds to -ln(Likelihood).
//Plain Gaussian centred at the origin. Valid for any number of dimensions.  Minimum value is the number of dimensior
double gauss(double params[], const int param_dim, int *fcall, bool *quit, const bool validvector, void** context)
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result += params[i]*params[i] + 1.0;
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  // Assign the numerical values chosen by Diver to variables of your choosing (this is optional,
  // but it is better e.g. to work with names like lambda1 for a coupling than params[0])
  freePar1 = params[0];
  freePar2 = params[1];
  freePar3 = params[2];
  freePar4 = params[3];
  freePar5 = params[4];

  // Create text file with the input for micromegas
  // ...
  // System call to micromegas
  // System call to DDCalc
  // Read from the output text file the observables and chi-squares, e.g.
  double massHiggs; // Higgs mass
  double Omega2;    // relic
  double DMmass;    // DM mass
  double sigmaP;    // DM-proton x-sect
  double chi2Higgs; // chi-square higgs
  double chi2Relic; // chi-square relic
  double chi2XENON; // chi-square XENON1T
  fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Relic,chi2XENON);

  // compopsite chi-square is the sum
  double chiComposite = chi2Higgs + chi2Relic + chi2XENON;

  // save observables for this point in parameter space
  params[5]  = massHiggs;
  ...
  ...
  params[11] = chi2XENON;
  params[12] = chiComposite;

  // return the value of the composite chi-square for this point in parameter space
  result     = chiComposite;

  *fcall += 1;
  *quit = false;
  return result;
}
```

Inserting the chi2BB code

- Diver chooses values for the free parameters and puts these values in the array ‘params’; these values constitute 1 point in parameter space
- In this example we have 5 free parameters, and we are assigning these values to new variables named freePar1, etc

```
//
// .....
//
//

// Assign the numerical values chosen by D
// but it is better e.g. to work with name
freePar1 = params[0];
freePar2 = params[1];
freePar3 = params[2];
freePar4 = params[3];
freePar5 = params[4];

// Create text file with the input for mic
// ...
// System call to micromegas
// System call to DDCalc
// Read from the output text file the obse
double massHiggs; // Higgs mass
```

Inserting the chi2BB code

- Our task is to tell Diver what value of the composite chi-square this point in parameter space has

```
43 //Function to be minimized.  Corresponds to -ln(Likelihood).
44 //Plain Gaussian centred at the origin. Valid for any number of dimensions.  Minimum value is the number of dimensior
45 double gauss(double params[], const int param_dim, int *fcall, bool *quit, const bool validvector, void** context)
46 {
47   double result = 0.0;
48   //for (int i = 0; i<param_dim; i++) result += params[i]*params[i] + 1.0;
49   if (!validvector)
50   {
51   result = DBL_MAX;
52   *fcall += 1;
53   *quit = false;
54   }
55
56   // Insert here code for chi2BB
57   //
58   // ......
59   //
60   //
61
62   // Assign the numerical values chosen by Diver to variables of your choosing (this is optional,
63   // but it is better e.g. to work with names like lambda1 for a coupling than params[0])
64   freePar1 = params[0];
65   freePar2 = params[1];
66   freePar3 = params[2];
67   freePar4 = params[3];
68   freePar5 = params[4];
69
70   // Create text file with the input for micromegas
71   // ...
72   // System call to micromegas
73   // System call to DDCalc
74   // Read from the output text file the observables and chi-squares, e.g.
75   double massHiggs; // Higgs mass
76   double Omega2;    // relic
77   double DMmass;    // DM mass
78   double sigmaP;    // DM-proton x-sect
79   double chi2Higgs; // chi-square higgs
80   double chi2Relic; // chi-square relic
81   double chi2XENON; // chi-square XENON1T
82   fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Relic,chi2XENON);
83
84   // compopsite chi-square is the sum
85   double chiComposite = chi2Higgs + chi2Relic + chi2XENON;
86
87   // save observables for this point in parameter space
88   params[5]  = massHiggs;
89   ...
90   ...
91   params[11] = chi2XENON;
92   params[12] = chiComposite;
93
94   // return the value of the composite chi-square for this point in parameter space
95   result     = chiComposite;
96
97   *fcall += 1;
98   *quit = false;
99   return result;
100 }
```

Inserting the chi2BB code

- To do that we create a text file with the values of the free parameters
- Next, we make a system call to micromegas passing this text file
- Next, we make a system call to DDCalc which reads the DM-proton x-section calculated by micromegas and computes the XENON1T chi-square
- Then we read from the output text file the values of the observables and the chi-squares and assign them to respective variables

```
freePar2 = params[1];
freePar3 = params[2];
freePar4 = params[3];
freePar5 = params[4];

// Create text file with the input for micromegas
// ...
// System call to micromegas
// System call to DDCalc
// Read from the output text file the observables and chi-squ
double massHiggs; // Higgs mass
double Omega2;    // relic
double DMmass;    // DM mass
double sigmaP;    // DM-proton x-sect
double chi2Higgs; // chi-square higgs
double chi2Relic; // chi-square relic
double chi2XENON; // chi-square XENON1T
fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Reli

// compopsite chi-square is the sum
double chiComposite = chi2Higgs + chi2Relic + chi2XENON;

// save observables for this point in parameter space
params[5]  = massHiggs;
...
...
params[11] = chi2XENON;
params[12] = chiComposite;
```

Inserting the chi2BB code

- The composite chi-square is just the sum of the 3 chi-squares computed

```
//Function to be minimized.  Corresponds to -ln(Likelihood).
//Plain Gaussian centred at the origin. Valid for any number of dimensions.  Minimum value is the number of dimensior
double gauss(double params[], const int param_dim, int *fcall, bool *quit, const bool validvector, void** context)
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result += params[i]*params[i] + 1.0;
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  // Assign the numerical values chosen by Diver to variables of your choosing (this is optional,
  // but it is better e.g. to work with names like lambda1 for a coupling than params[0])
  freePar1 = params[0];
  freePar2 = params[1];
  freePar3 = params[2];
  freePar4 = params[3];
  freePar5 = params[4];

  // Create text file with the input for micromegas
  // ...
  // System call to micromegas
  // System call to DDCalc
  // Read from the output text file the observables and chi-squares, e.g.
  double massHiggs; // Higgs mass
  double Omega2;    // relic
  double DMmass;    // DM mass
  double sigmaP;    // DM-proton x-sect
  double chi2Higgs; // chi-square higgs
  double chi2Relic; // chi-square relic
  double chi2XENON; // chi-square XENON1T
  fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Relic,chi2XENON);

  // compopsite chi-square is the sum
  double chiComposite = chi2Higgs + chi2Relic + chi2XENON;

  // save observables for this point in parameter space
  params[5]  = massHiggs;
  ...
  ...
  params[11] = chi2XENON;
  params[12] = chiComposite;

  // return the value of the composite chi-square for this point in parameter space
  result     = chiComposite;

  *fcall += 1;
  *quit = false;
  return result;
}
```

Inserting the chi2BB code

- The composite chi-square is just the sum of the 3 chi-squares computed

```
// ...
// System call to micromegas
// System call to DDCalc
// Read from the output text file the observables and chi-squares, e.g.
double massHiggs; // Higgs mass
double Omega2;    // relic
double DMmass;    // DM mass
double sigmaP;    // DM-proton x-sect
double chi2Higgs; // chi-square higgs
double chi2Relic; // chi-square relic
double chi2XENON; // chi-square XENON1T
fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Relic,chi2XENON

// compopsite chi-square is the sum
double chiComposite = chi2Higgs + chi2Relic + chi2XENON;

// save observables for this point in parameter space
params[5]  = massHiggs;
...
...
params[11] = chi2XENON;
params[12] = chiComposite;

// return the value of the composite chi-square for this point in parame
result     = chiComposite;

*fcall += 1;
*quit = false;
return result;
```

Inserting the chi2BB code

- Next, we must save the values of the observables and the chi-squares

```
//Function to be minimized.  Corresponds to -ln(Likelihood).
//Plain Gaussian centred at the origin. Valid for any number of dimensions.  Minimum value is the number of dimensior
double gauss(double params[], const int param_dim, int *fcall, bool *quit, const bool validvector, void** context)
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result += params[i]*params[i] + 1.0;
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  // Assign the numerical values chosen by Diver to variables of your choosing (this is optional,
  // but it is better e.g. to work with names like lambda1 for a coupling than params[0])
  freePar1 = params[0];
  freePar2 = params[1];
  freePar3 = params[2];
  freePar4 = params[3];
  freePar5 = params[4];

  // Create text file with the input for micromegas
  // ...
  // System call to micromegas
  // System call to DDCalc
  // Read from the output text file the observables and chi-squares, e.g.
  double massHiggs; // Higgs mass
  double Omega2;    // relic
  double DMmass;    // DM mass
  double sigmaP;    // DM-proton x-sect
  double chi2Higgs; // chi-square higgs
  double chi2Relic; // chi-square relic
  double chi2XENON; // chi-square XENON1T
  fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Relic,chi2XENON);

  // compopsite chi-square is the sum
  double chiComposite = chi2Higgs + chi2Relic + chi2XENON;

  // save observables for this point in parameter space
  params[5]  = massHiggs;
  ...
  ...
  params[11] = chi2XENON;
  params[12] = chiComposite;

  // return the value of the composite chi-square for this point in parameter space
  result     = chiComposite;

  *fcall += 1;
  *quit = false;
  return result;
}
```

Inserting the chi2BB code

- These have to be saved in the same array ‘params’, but next to the free parameters, i. e. starting from params[5] onwards in this example

```
// compopsite chi-square is the su
double chiComposite = chi2Higgs +

// save observables for this point
params[5]  = massHiggs;
...
...
params[11] = chi2XENON;
params[12] = chiComposite;

// return the value of the composi
result     = chiComposite;

*fcall += 1;
*quit = false;
```

Inserting the chi2BB code

- Finally, we return to Diver the value of the composite chi-square

```
//Function to be minimized.  Corresponds to -ln(Likelihood).
//Plain Gaussian centred at the origin. Valid for any number of dimensions.  Minimum value is the number of dimensior
double gauss(double params[], const int param_dim, int *fcall, bool *quit, const bool validvector, void** context)
{
  double result = 0.0;
  //for (int i = 0; i<param_dim; i++) result += params[i]*params[i] + 1.0;
  if (!validvector)
  {
  result = DBL_MAX;
  *fcall += 1;
  *quit = false;
  }

  // Insert here code for chi2BB
  //
  // ......
  //
  //

  // Assign the numerical values chosen by Diver to variables of your choosing (this is optional,
  // but it is better e.g. to work with names like lambda1 for a coupling than params[0])
  freePar1 = params[0];
  freePar2 = params[1];
  freePar3 = params[2];
  freePar4 = params[3];
  freePar5 = params[4];

  // Create text file with the input for micromegas
  // ...
  // System call to micromegas
  // System call to DDCalc
  // Read from the output text file the observables and chi-squares, e.g.
  double massHiggs; // Higgs mass
  double Omega2;    // relic
  double DMmass;    // DM mass
  double sigmaP;    // DM-proton x-sect
  double chi2Higgs; // chi-square higgs
  double chi2Relic; // chi-square relic
  double chi2XENON; // chi-square XENON1T
  fscanf(....,massHiggs,Omega2,DMmass,sigmaP,chi2Higgs,chi2Relic,chi2XENON);

  // compopsite chi-square is the sum
  double chiComposite = chi2Higgs + chi2Relic + chi2XENON;

  // save observables for this point in parameter space
  params[5]  = massHiggs;
  ...
  ...
  params[11] = chi2XENON;
  params[12] = chiComposite;

  // return the value of the composite chi-square for this point in parameter space
  result     = chiComposite;

  *fcall += 1;
  *quit = false;
  return result;
}
```

Inserting the chi2BB code

- Finally, we return to Diver the value of the composite chi-square
- This completes the code for the chi2BB
- Next, we compile the program with the ‘make’ instruction as before and execute it …

```
...
...
params[11] = chi2XENON;
params[12] = chiComposite;

// return the value of the compos
result     = chiComposite;

*fcall += 1;
*quit = false;
return result;
}
```

Execute, wait a lot of time and plot

- In this example we execute Diver with 4 cores,
- ‘name_of_executable’ is the program that results from compiling the code just shown
- We redirect the output of the program to the text file ‘log.txt’ and we run in the ‘background’

```
mpiexec -np 4 name_of_executable >> log.txt &
```

```
git clone https://github.com/patscott/pippi.git
sudo apt-get -y install ctioga2
```

```
~/pippi/pippi  q4.pip
```

Execute, wait a lot of time and plot

- After a while (which can be of the order of ~1-2 months!) Diver generates a file with the results
- To parse this file, we need the tool called 'pippi'
- Pippi uses ctioga2 to plot so we also install this
- We call pippi by means of a configuration file, in this example the file called 'q4.pip', basically this files just defines which of observables we wish to plot

```
mpiexec -np 4 name_of_executable >> log.txt &



git clone https://github.com/patscott/pippi.git
sudo apt-get -y install ctioga2


~/pippi/pippi  q4.pip
```

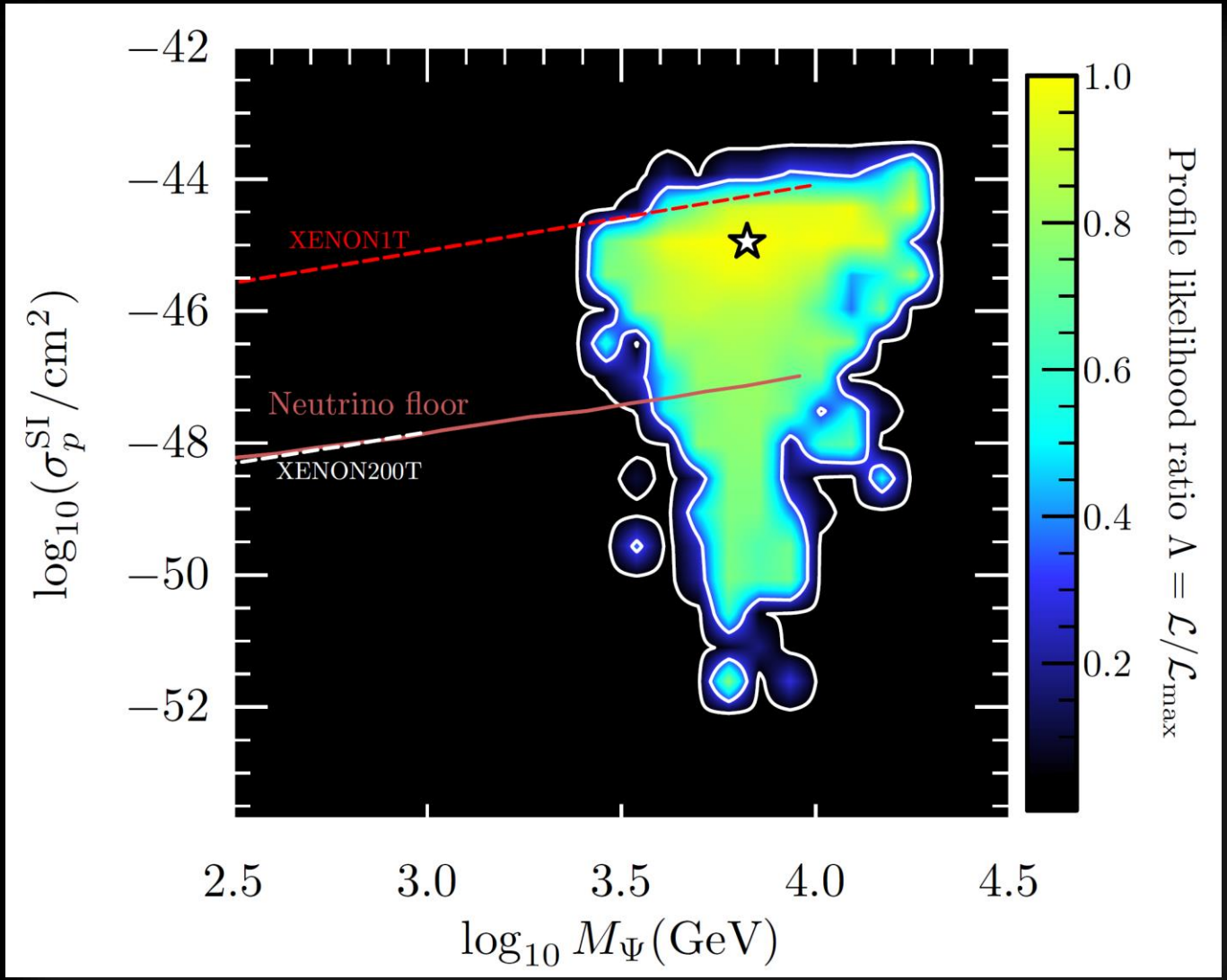
Execute, wait a lot of time and plot

- This is an example of a pippi configuration file to plot the DM-proton x-section as a function of the DM mass
- But I will not enter into details of this here …

```
-----------------------------
---Common fields--------------          ;EFN = empty for none

main_chain = './res/q4.sam'            ;Filename of chain to operate on.  For a group in an hdf5 file, something like 'myfile.hdf5:/group'
comparison_chain =                     ;Filename of secondary chain to operate on, for comparison (EFN)

do_posterior_pdf = F                   ;Parse, script or plot posterior pdfs
do_profile_like = T                    ;Parse, script or plot chain into profile likelihoods
oneD_contour_levels =                  ;Contour levels to plot in 1D plots (EFN; script operation ignores actual values)
twoD_contour_levels =  68.3 95.4       ;Contour levels to plot in 2D plots (EFN; script operation ignores actual values)
oneD_plot_quantities =                 ;Parameters/observables to construct 1D plots of (EFN)
twoD_plot_quantities = {11 12};        ;Parameters/observable combinations to make 2D plots of (EFN)
plot_observables =                     ;Observables to make 2D plots of (EFN)

-----------------------------
---Parsing--------------------

parse_dir = 'parse_diver'              ;Directory to save parsing output in (EFN; default is to place parsed data in same folder as chains)

cut_on_invalid_observables = F         ;Cut on any invalid entries in an hdf5 file, not just the likelihood (defaults to false, meaning cut on likelihood only)

default_bins = 40   ;                  ;Default number of bins to sort samples into in each direction.
number_of_bins =                       ;Bins to use for specific observables (overrides default)
interpolated_resolution = 4000 ;       ;Resolution of binwise interpolation for plotting (~300+ for publication)
interpolation_method = 'bilinear'      ;Either bilinear (default) or spline (watch out for ringing in the latter case)

chain_type = other                     ;Algorithm used to generate chain (valid: MCMC, MultiNest, other)
compute_evidence = F                   ;Compute and print evidence (only if chain_type = MCMC)
bf_lnlike_for_profile_like =           ;Denominator of profile likelihood ratio (if empty, set to best fit determined from chain)

use_log_scale =  11 12                 ;Parameters/observables that need to be converted to a log scale for display (EFN)
quantity_rescalings =                  ;Scaling factors to apply to any pippi datastream indices (EFN)
data_ranges =                          ;Discard any points outside these ranges of parameters/observables 25:{-40.,1.}

labels_from_file =                     ;Either a filename or blank (quantity_labels is ignored if a filename is given)
preamble = 'from preamble_example import *' ;A one-line python preamble to run before processing functional datastreams.
assign_to_pippi_datastream =           ;Assign a python function or named hdf5 data stream to a particular pippi datastream index (use pippi probe to test; EFN)
quantity_labels =  1:'-lnlike'        \
                   4:'freePar1'       \
                   5:'freePar2'       \
                   6:'freePar3'       \
                   7:'freePar4'       \
                   8:'freePar5'       \
                   9:'$m_h$'          \
                  10:'$\Omega h^2$'   \
                  11:'$\log_{10} M_\Psi \textrm{(GeV)}$'                \
                  12:'$\log_{10} (\sigma_p^{\rm SI} / \textrm{cm}^2)$'   \
                  13:'$\chi^2_h$'     \
                  14:'$\chi^2_{\Omega h^2}$'    \
                  15:'$\chi^2_{DD}$'    \
                                       ;labels to apply to different pippi datastream indices (LaTeX for all but multiplicity, prior and likelihood)
-----------------------------
---Scripting------------------

script_dir = 'scripts_diver'           ;Directory to save scripts in (EFN; default is to place scripts in same folder as parse output)

legend_on_1D =                         ;1D plots that should include legends
legend_locations_1D =                  ;Locations of legends for 1D plots (subset of twoD_plot_quantities)
plot_as_histograms_1D = F              ;Plot 1D distributions as histograms with no interpolation
key_on_1D =                            ;2D plots that should include keys for best fit, posterior mean and/or reference pt
key_locations_1D =                     ;Locations of keys for 2D plots

legend_on_2D =  ;                      ;2D plots that should include legends
legend_locations_2D = ;                ;Locations of legends for 2D plots (single location or partial list that will be filled in with 'bl')
key_on_2D =                            ;2D plots that should include keys for best fit, posterior mean and/or reference pt
```

Execute, wait a lot of time and plot

- Suffice to say that it generates this plot of the DM-proton x-section as a function of the DM mass



Subtleties

- There are certain subtleties with Diver and Pippi which are hard to explain in a talk
- We will address these topics with a lot more of detail in the corresponding tutorials
- We shall also cover Micromegas, DDCalc and others in respective tutorials

Thank you for your attention!