Tool for High Energy Physics

Lecture 2
From tree-level to one-loop calculations

Rui Santos

ISEL & CFTC

7 September 2011

Rules for building the Lagrangian

$$f(x) = \int_1^{+\infty} \frac{1}{x+y}\,dy$$

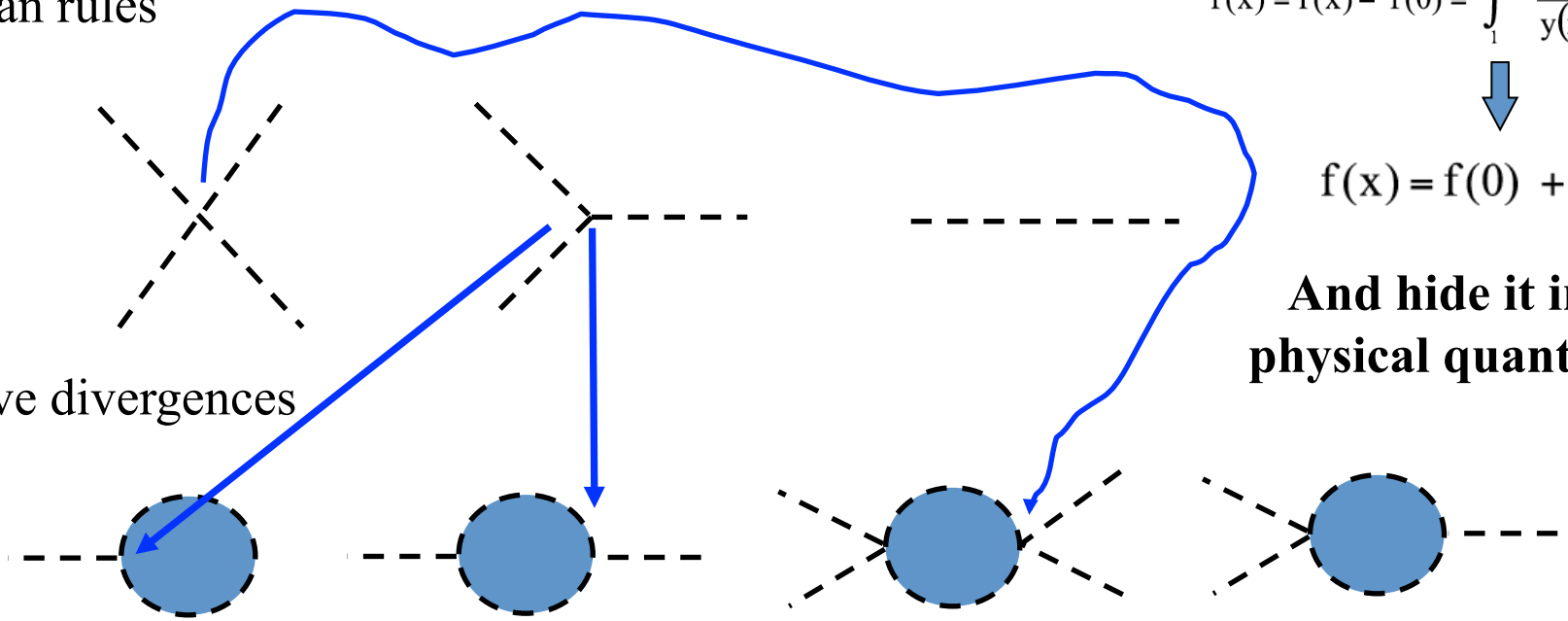
$$\bar{f}(x) = f(x) - f(0) = \int_1^{+\infty} \frac{-x}{y(x+y)}\,dy$$

$$f(x) = f(0) + \bar{f}(x)$$

Feynman rules

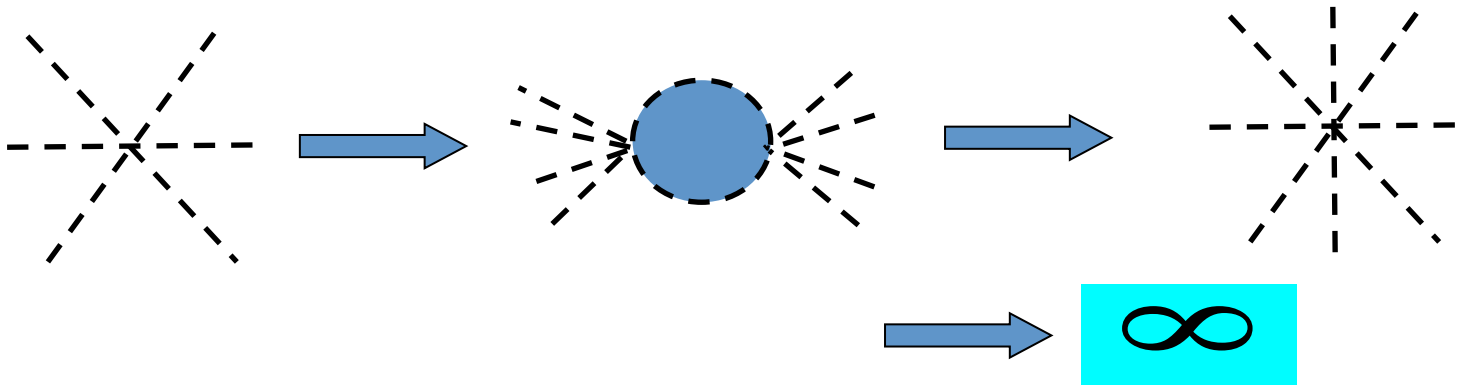
And hide it in physical quantity

Primitive divergences



Stop at H^4

Higher order terms



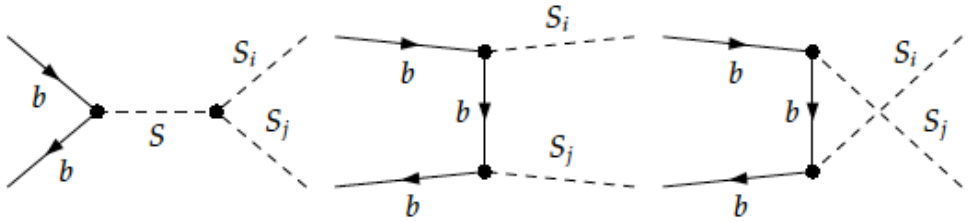
Why do we calculate beyond leading order?

a) Because higher order processes dominate

$pp \to S_i S_j$ Scalar pair production in a 2HDM extension of the SM

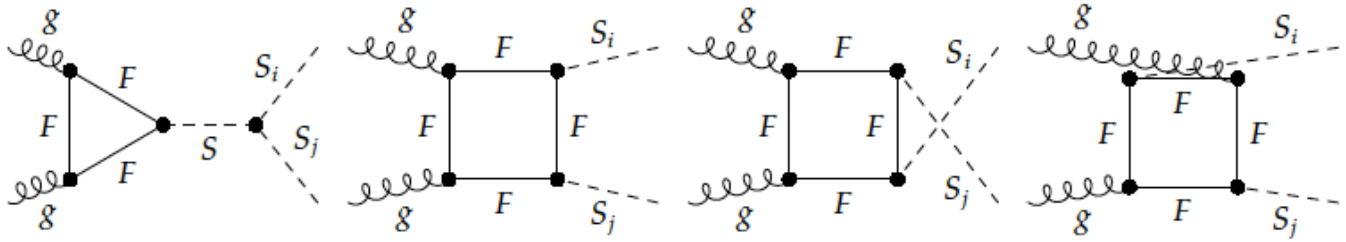
Two contributions at the parton level

$q\bar{q} \to S_i S_j$



Scalar-fermions coupling proportional to the fermion mass – only q=b considered.

$gg \to S_i S_j$



b) Because experimental precision force us to improve the calculation's precision

Calculate $q\bar{q} \to S_i S_j$ at 1-loop level

How do we calculate beyond leading order?

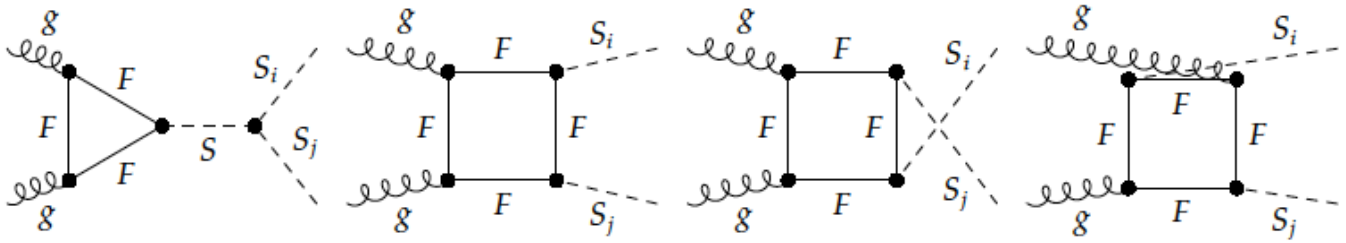
We find a systematic way to hide all infinities and get finite results for the cross sections – this is called renormalization

$$M_W^2 \to M_W^2 + \delta M_W^2$$

$$W_{\mu 0}^{\pm} = Z_W^{1/2} W_\mu^{\pm}$$

We generate the counter-term Lagrangian by redefining the fields and couplings in the original Lagrangian. We now use this new Lagrangian together with a recipe to discard the infinities (on-shell, Minimal Subtraction,…).

However, when the process we want to calculate does not exist at tree-level, there is no term in the Lagrangian to generate the counterterms. Therefore the sum of all the infinities coming from the different diagrams has to cancel.



So, these lectures do not include learning how to use counter-terms in FeynArts/ FormCalc/LoopTools.

Tools for calculations beyond leading order

FeynArts

FormCalc

LoopTools

FeynArts is a Mathematica package for the generation and visualization of Feynman diagrams and amplitudes. Its main features are:

The generation of diagrams is possible at three levels: generic fields, classes of fields, or specific particles.

The model information is contained in two special files: The generic model file defines the representation of the kinematical quantities like spinors or vector fields. The classes model file sets up the particle content and specifies the actual couplings. Since the user can create own model files, the applicability of FeynArts is virtually unlimited within perturbative quantum field theory. As generic model the Lorentz formalism (Lorentz.gen) and as classes model the electroweak Standard Model in several variations (SM.mod, SMQCD.mod, SMbgf.mod) and the MSSM (MSSM.mod, MSSMQCD.mod) are supplied.

In addition to ordinary diagrams, FeynArts can generate counter-term diagrams and diagrams with placeholders for one-particle irreducible vertex functions.

Tools for calculations beyond leading order

FeynArts

FormCalc

LoopTools

FormCalc is a Mathematica package for the calculation of tree-level and one-loop Feynman diagrams. It reads diagrams generated with FeynArts and returns the results in a way well suited for further numerical and analytical evaluation. FormCalc can in fact write out a complete Fortran subroutine to compute the squared matrix element for a given process. In addition to a comprehensive manual, several demo calculations are included in the FormCalc distribution, which show how the programs are used.

The following simplifications are performed by FormCalc:

-indices are contracted as far as possible,
- fermion traces are evaluated,
- open fermion chains are simplified using the Dirac equation,
- colour structures are simplified using the SU(N) algebra,
- the tensor reduction is done,
- the results are partially factored,
- abbreviations are introduced.

Tools for calculations beyond leading order

FeynArts

FormCalc

LoopTools

LoopTools is a package for evaluation of scalar and tensor one-loop integrals based on the FF package by G.J. van Oldenborgh. It features an easy Fortran, C++, and Mathematica interface to the scalar one-loop functions of FF and in addition provides the 2-, 3-, and 4-point tensor coefficient functions.

http://www.feynarts.de/

The automatic installation script gets you started quickly and easily

- Download the shell script FeynInstall [5 kB, MD5: ac79b3a980ec752794804e537dc2da3c]. (Use the right mouse button and "Save Link As...")
- Make it executable with chmod 755 FeynInstall.
- Run it in the directory in which you want the packages installed: ./FeynInstall.
- The script separately prompts you for the installation of FeynArts, FormCalc, and LoopTools.
- Finally, it asks whether to include FeynArts and FormCalc in Mathematica's $Path.

Generating the Feynman Rules for FeynArts

First we go back to LanHEP to generate the Feynman rules (done)

Feynman Rules for FeynArts → ./lhep -feynarts SEDunbroken.mdl

model10.mod
model10.h
mdl_ini10.F ← Do we have to modify these files?
model10.gen

```
subroutine ModelConstIni(*)
implicit none
```

→

```
subroutine ModelConstIni(teste)
implicit none
integer teste
```

```
subroutine ModelVarIni(sqrtS, *)
implicit none
double precision sqrtS
double precision Alfas
```

→

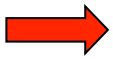
```
subroutine ModelVarIni(teste,sqrtS)
implicit none
double precision sqrtS
double precision Alfas
integer teste
```

Not a universal rule – tools are updated independently

Generating the Feynman Rules for FeynArts

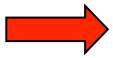
This is a universal rule

model10.mod
model10.gen



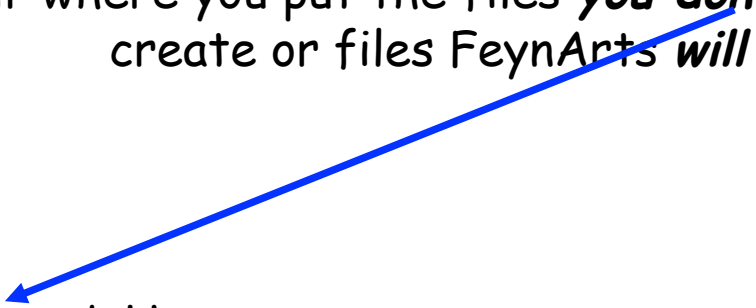
Include in /……/FeynArts/Model

model10.h
mdl_ini10.F



Include in the G*/Feyn/drivers dir

A dir where you put the files ***you don't want*** FeynArts to create or files FeynArts ***will not create***



run.f and model.h

model10.h
mdl_ini10.F

This is already done

$AA \to AA$

➡ Building the Mathematica file AAAA.m

```
(* AA --> AA in SEDunbroken *)

Needs["FeynArts`FeynArts`"]
Needs["FormCalc`FormCalc`"]
```

```
time1 = SessionTime[]
```

```
process = {V[1], V[1]} ->  {V[1], V[1]}

name = "AAAA"

SetOptions[InsertFields, Model -> "model10",GenericModel->"model10"]
```

```
SetOptions[Paint, PaintLevel -> {Generic,Classes}, ColumnsXRows -> {4, 5}]

(* take the comments out if you want the diagrams painted*)
$PaintSE = MkDir[name <> ".diagrams"];
DoPaint[diags_, file_, opt___] := Paint[diags, opt,
  DisplayFunction -> (Export[ToFileName[$PaintSE, file <> ".ps"], #]&)]

Print["Self energies"]

tops = CreateTopologies[1, 2 -> 2, SelfEnergiesOnly];
ins = InsertFields[tops, process];
DoPaint[ins, "self"];
self = CalcFeynAmp[CreateFeynAmp[ins]];
```

```
Print["Vertices"]

tops = CreateTopologies[1, 2 -> 2, TrianglesOnly];
ins = InsertFields[tops, process];
DoPaint[ins, "vert"];
vert = CalcFeynAmp[CreateFeynAmp[ins]];
```

$AA \to AA$

➡ Building the Mathematica file AAAA.m

```
Print["Boxes"]

tops = CreateTopologies[1, 2 -> 2, BoxesOnly];
ins = InsertFields[tops, process];
DoPaint[ins, "box"];
box = CalcFeynAmp[CreateFeynAmp[ins]];

amps = {self, vert, box}
```

```
abbr = OptimizeAbbr[Abbr[]]

col = ColourME[All,All]

subexpr = OptimizeAbbr[Subexpr[]]

dir = SetupCodeDir[name <> ".fortran", Drivers -> name <> ".drivers"]

WriteSquaredME[{}, {self,vert,box},col, abbr,subexpr, dir]

WriteRenConst[amps, dir]
```

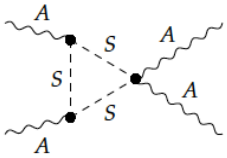
```
Print["time used: ", SessionTime[] - time1]
```

➡ Running the Mathematica file ➡

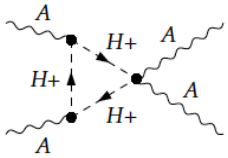
math
<<AAAA.m
Exit

Go to AAAA.fortran
./configure
make

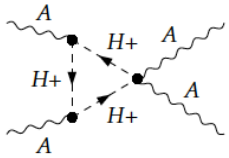
$AA \to AA$



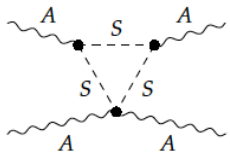
T1 G1 N1



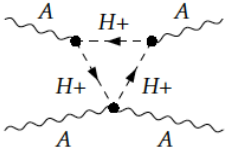
T1 C1 N2



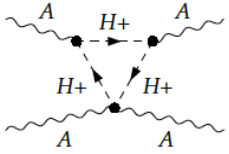
T1 C2 N3



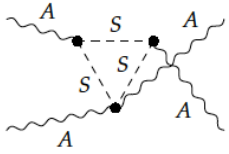
T2 G1 N4



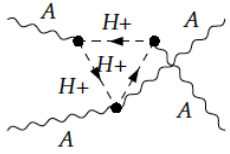
T2 C1 N5



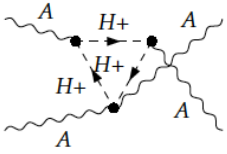
T2 C2 N6



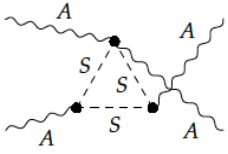
T3 G1 N7



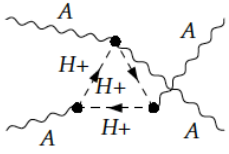
T3 C1 N8



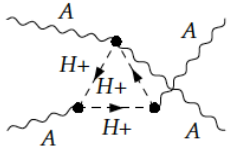
T3 C2 N9



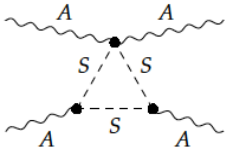
T4 G1 N10



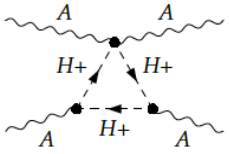
T4 C1 N11



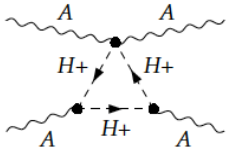
T4 C2 N12



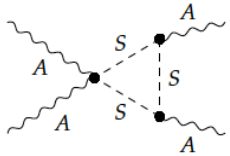
T5 G1 N13



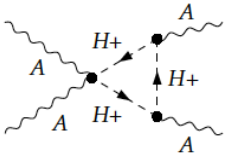
T5 C1 N14



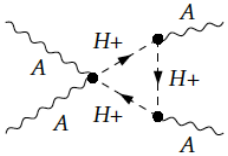
T5 C2 N15



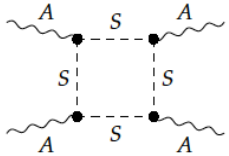
T6 G1 N16



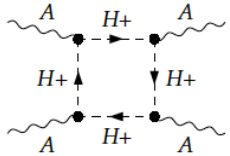
T6 C1 N17



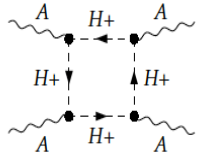
T6 C2 N18



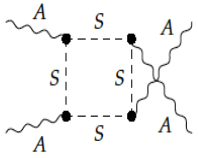
T7 G1 N19



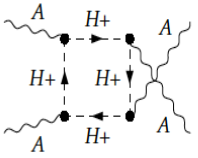
T7 C1 N20



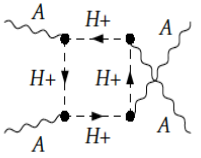
T7 C2 N21



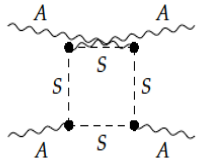
T8 G1 N22



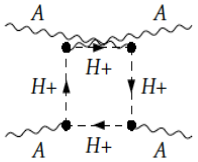
T8 C1 N23



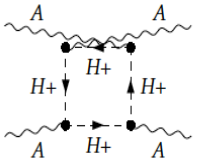
T8 C2 N24



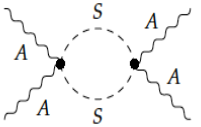
T9 G1 N25



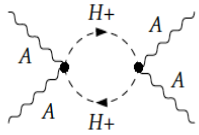
T9 C1 N26



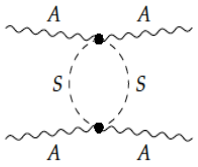
T9 C2 N27



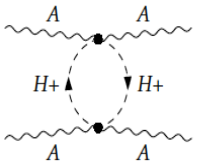
T10 G1 N28



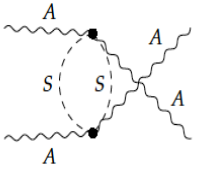
T10 C1 N29



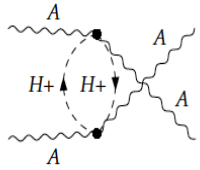
T11 G1 N30



T11 C1 N31



T12 G1 N32



T12 C1 N33

$$AA \to AA$$

```
* process.h
* defines all process-dependent parameters

      #define TYPE1 VECTOR
      #define MASS1 0
      #define CHARGE1 0

      #define TYPE2 VECTOR
      #define MASS2 0
      #define CHARGE2 0

      #define TYPE3 VECTOR
      #define MASS3 0
      #define CHARGE3 0

      #define TYPE4 VECTOR
      #define MASS4 0
      #define CHARGE4 0

* The combinatorial factor for identical particles in the final state:
* 1/n! for n identical particles, 1 otherwise

#define IDENTICALFACTOR 1

 * Possibly a colour factor, e.g.
 * - an additional averaging factor if any of the incoming particles
 *   carry colour,
 * - the overall colour factor resulting from the external particles
 *   if that cannot computed by FormCalc (e.g. if the model has no
 *   colour indices, as SMew.mod).

 #define COLOURFACTOR 1

* The scale at which the interaction takes place
* (= the factorization scale for an hadronic process).

#define SCALE sqrtS
```

$AA \to AA$

```
* process.h
* defines all process-dependent parameters

* NCOMP is the number of components of the result vector.  Currently
* the components are 1 = tree-level result, 2 = one-loop result.

#define NCOMP 2

* Choose the appropriate luminosity for the collider:
* - lumi_parton.F for a "parton collider" (e.g. e+ e- -> X),
* - lumi_hadron.F for a hadron collider (e.g. p pbar -> X),
* - lumi_photon.F for a photon collider (gamma gamma -> X)

#define LUMI "lumi_parton.F"

* for lumi_hadron.F: PARTON1 and PARTON2 identify the
* incoming partons by their PDG code, where
* 0 = gluon
* 1 = down   3 = strange   5 = bottom
* 2 = up     4 = charm     6 = top

#define PARTON1 1
#define PARTON2 1
#define PDFSET "cteq5l.LHgrid"
#define PDFMEM 0

* Include the kinematics-dependent part.

#include "2to2.F"
```

Next file run.F

$AA \to AA$

```
* run.F
* defines parameter settings for one run, then includes main.F
* Whether to run in debugging mode

#define DEBUG

* Uncomment the following to check UV- and IR-finiteness

c#define DELTA 1D7
c#define MUDIM 1D100
c#define LAMBDA 1D10
 * The following sets all ren. constants to zero for debugging
 * or if calculating a tree-level reaction.

 c#define NO_RENCONST
* options for model_thdm.F:
c#define MODEL_TYPE_I
c#define MODEL_TYPE_II
c#define NO_EXCLUSION_LIMITS

#define MODEL "mdl_ini10.F"

 * The LOOPn set up the model parameters for the calculation, either
 * - as fixed declarations, e.g.
 *       MH = 115
 * - or as do-loops terminating on label 1, e.g.
 *       do 1 MH = 100, 500, 10

 #define LOOP1
 #define LOOP2
 #define LOOP3
```

$AA \to AA$

```
* The multidimensional integration routines also need

#define METHOD DIVONNE
#define VERBOSE 1
#define MINEVAL 0
#define MAXEVAL 50000

* for Vegas:
#define NSTART 1000
#define NINCREASE 500

* for Suave:
#define NNEW 1000
#define FLATNESS 50

* for Divonne:
#define KEY1 47
#define KEY2 1
#define KEY3 1
#define MAXPASS 5
#define BORDER 1D-6
#define MAXCHISQ 10
#define MINDEVIATION .25D0

* for Cuhre:
#define KEY 0

* Now go for it:

#include "main.F"
```

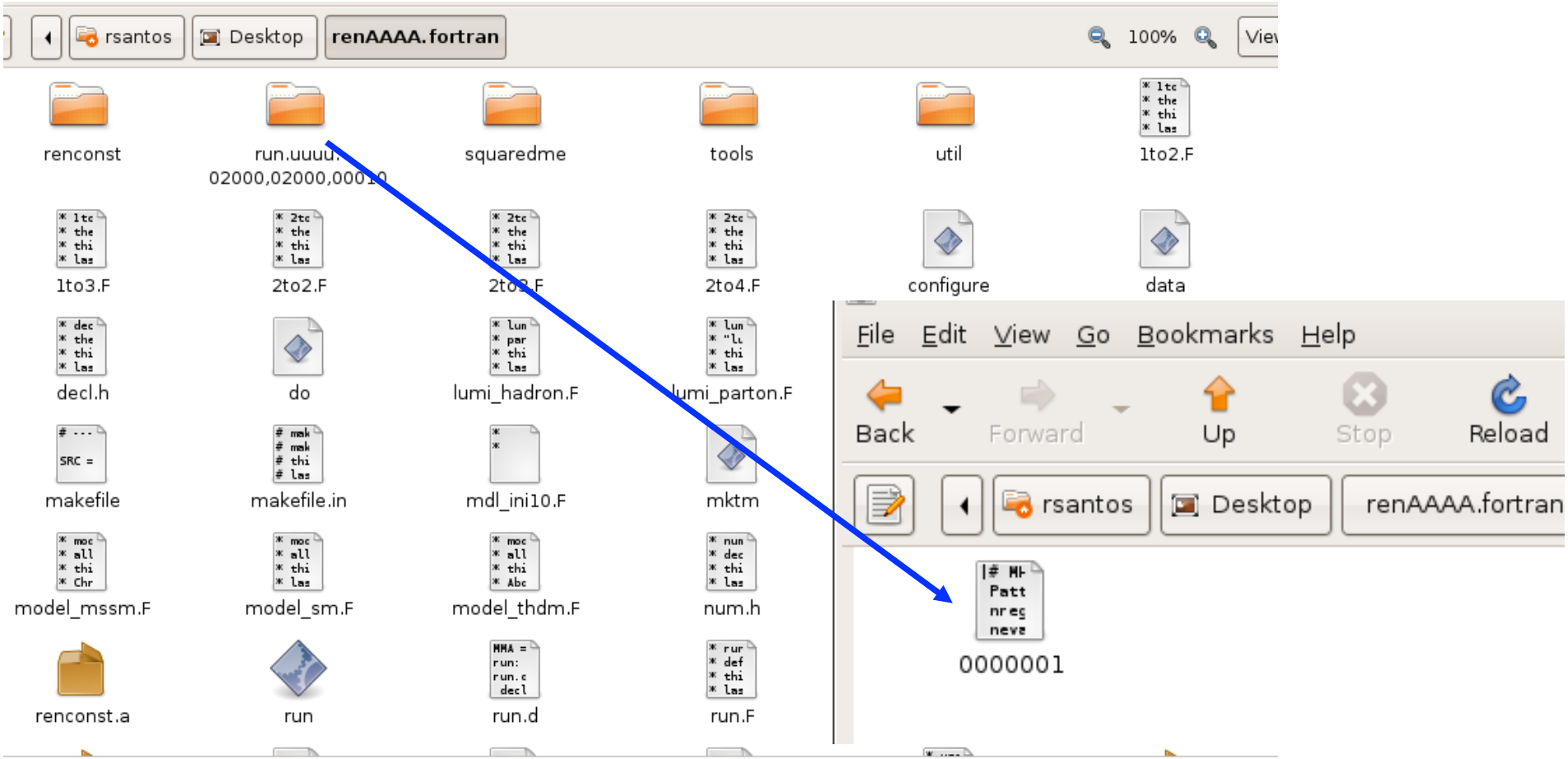
$AA \to AA$

Now we are ready to get results **(which we will not!)**

1 Move to the Fortran directory

2 ./configure

3 make run.F (**the executable file would be created and would be named run**)



$AA \to AA$

New Open Save | Print... | Undo Redo | Cut Copy Paste | Find Replace

run.F | gagahhdlim1.gpl | 0000001

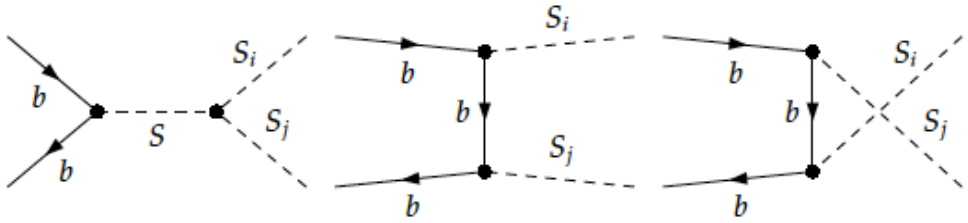
```
# MHc=  100.0000
 Patterson integration results:
 nregions = 1
 neval    = 43
 fail     = 0
 1    0.00000000000000     +-      0.00000000000000         p = -1.000
 2   0.3788536497624 51E-07 +-    0.1815376692586 87E-12     p = -1.000
      2000.000000
+       0.00000000000000       0.3788536497624 51E-07
+       0.00000000000000       0.1815376692586 87E-12
```

The 2HDM

$pp \to S_i S_j$ Scalar pair production in a 2HDM extension of the SM

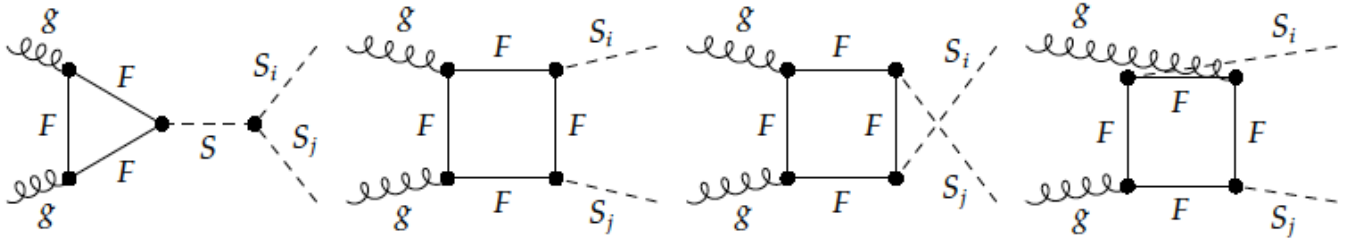
Two contributions at the parton level

$q\bar{q} \to S_i S_j$



Scalar-fermions coupling proportional to the fermion mass – only q=b considered.

$gg \to S_i S_j$

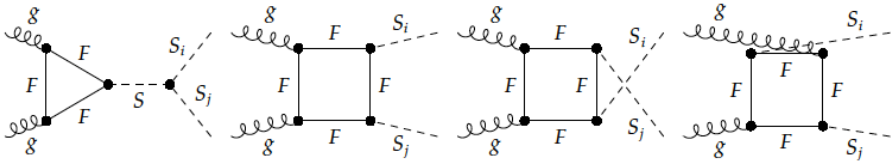


Only generic diagrams are shown.

The 2HDM

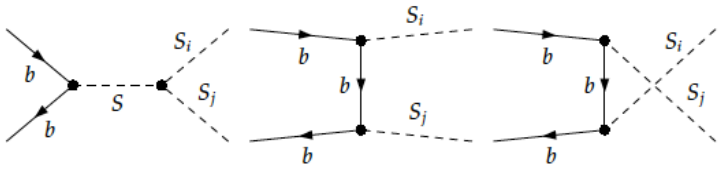
➡ Building the Mathematica files

```
(* gg --> hh in THDM *)

<<FeynArts`FeynArts`
<<FormCalc`FormCalc`

process = {V[5], V[5]} ->  {S[1], S[1]}

name = "gghh-THDM"

SetOptions[InsertFields, Model -> "THDMQCD"]
```



{F[4,{3}],-F[4,{3}]}

name = "treebbhh-THDM"




```
InitializeModel[THDMQCD];
RenConst[Gammah1]:=ImTilde[SelfEnergy[S[1]->S[1],Mh0]]/Mh0;
RenConst[Gammah2]:=ImTilde[SelfEnergy[S[2]->S[2],Mhh]]/Mhh;

WriteRenConst[{Gammah1,Gammah2}, dir]
```

➡ Changing the model file to include constraints model_thdm.F

```
**************************************************************************

        subroutine ModelConstIni(fail)
        implicit none
        integer fail
------------------------------------------------------------------------------
      if( abs(MA02 - MHH2) .gt. 1D-13 .and.
     &    abs(MHp2 - MHH2) .gt. 1D-13 ) then
        DeltaRho = GF/(8*sqrt2*pi**2)*(
     &    MHp2*(1 - MA02/(MHp2 - MA02)*log(MHp2/MA02)) +
     &    CBA**2*Mh02*(MA02/(MA02 - Mh02)*log(MA02/Mh02) -
     &                 MHp2/(MHp2 - Mh02)*log(MHp2/Mh02)) +
     &    SBA**2*MHH2*(MA02/(MA02 - MHH2)*log(MA02/MHH2) -
     &                 MHp2/(MHp2 - MHH2)*log(MHp2/MHH2)) )

        if( DeltaRho .gt. DeltaRho_UPPERBOUND ) then
          WARN "model_thdm: DeltaRho = ", DeltaRho,
     &      " > DeltaRho_UPPERBOUND = ", DeltaRho_UPPERBOUND
          fail = 3
          return
        endif
```

This variable is measured experimentally and can be updated in the beginning of the model file.

From the data to the plot