PyHEP 2024
“Python in HEP” Users Workshop (Online)
Jul 1-4, 2024

VLQcalc: a Python Module for Calculating Vector-like Quark Couplings

Ali Can Canbay, Orhan Cakir
Ankara University Department of Physics

VLQcalc

Based on Vector-like quarks (VLQ) models prepared with FeynRules [1].

Available models:

- VLQ_UFO [2]
- VLQ_v4_UFO [3]

Main features:

- Converting couplings between models
- Decay width and branching ratio calculations
- Optimization of couplings with given branching and Γ/m ratios
- Creating a Madgraph5 [4] input file

Architecture:

VLQcalc
- decays.py
- functions.py
- madgraph.py
- model.py
- modelConverter.py

decays.py : Contains the decay functions
functions.py : Includes coupling optimization functions for Γ/m
madgraph.py : It contains the MG5 object. Used to create MG5 input file
model.py : Contains the VLQ object. All modules except madgraph.py are used with the VLQ object.
modelConverter.py : Contains functions that convert couplings

VLQcalc GitHub page

Main Features: Converting couplings

$$\mathcal{L} = \kappa_T \left\{ \sqrt{\frac{\zeta_i \xi_W^T}{\Gamma_W^0}} \frac{g}{\sqrt{2}} [\bar{T}_{L/R} W_\mu^+ \gamma^\mu d_{L/R}^i] + \sqrt{\frac{\zeta_i \xi_Z^T}{\Gamma_Z^0}} \frac{g}{2c_W} [\bar{T}_{L/R} Z_\mu \gamma^\mu u_{L/R}^i] - \sqrt{\frac{\zeta_i \xi_H^T}{\Gamma_H^0}} \frac{M}{v} [\bar{T}_{R/L} H u_{L/R}^i] \right\}$$
$$+ \kappa_B \left\{ \sqrt{\frac{\zeta_i \xi_W^B}{\Gamma_W^0}} \frac{g}{\sqrt{2}} [\bar{B}_{L/R} W_\mu^- \gamma^\mu u_{L/R}^i] + \sqrt{\frac{\zeta_i \xi_Z^B}{\Gamma_Z^0}} \frac{g}{2c_W} [\bar{B}_{L/R} Z_\mu \gamma^\mu d_{L/R}^i] - \sqrt{\frac{\zeta_i \xi_H^B}{\Gamma_H^0}} \frac{M}{v} [\bar{B}_{R/L} H d_{L/R}^i] \right\}$$
$$+ \kappa_X \left\{ \sqrt{\frac{\zeta_i}{\Gamma_W^0}} \frac{g}{\sqrt{2}} [\bar{X}_{L/R} W_\mu^+ \gamma^\mu u_{L/R}^i] \right\} + \kappa_Y \left\{ \sqrt{\frac{\zeta_i}{\Gamma_W^0}} \frac{g}{\sqrt{2}} [\bar{Y}_{L/R} W_\mu^- \gamma^\mu d_{L/R}^i] \right\} + h.c..$$

VLQ_UFO

$\kappa_W = \kappa, \quad \kappa_Z = \tilde{\kappa}, \quad \kappa_H = \hat{\kappa}$

$$\mathcal{L}_{\text{VLQ}} = i\bar{Y}\not{D}Y - m_Y\bar{Y}Y + i\bar{B}\not{D}B - m_B\bar{B}B + i\bar{T}\not{D}T - m_T\bar{T}T + i\bar{X}\not{D}X - m_X\bar{X}X$$
$$- h\left[\bar{B}\left(\hat{\kappa}_L^B P_L + \hat{\kappa}_R^B P_R\right) q_d + \text{h.c.}\right]$$
$$- h\left[\bar{T}\left(\hat{\kappa}_L^T P_L + \hat{\kappa}_R^T P_R\right) q_u + \text{h.c.}\right]$$
$$+ \frac{g}{2c_W}\left[\bar{B}\not{Z}\left(\tilde{\kappa}_L^B P_L + \tilde{\kappa}_R^B P_R\right) q_d + \text{h.c.}\right]$$
$$+ \frac{g}{2c_W}\left[\bar{T}\not{Z}\left(\tilde{\kappa}_L^T P_L + \tilde{\kappa}_R^T P_R\right) q_u + \text{h.c.}\right]$$
$$+ \frac{\sqrt{2}g}{2}\left[\bar{Y}\not{W}\left(\kappa_L^Y P_L + \kappa_R^Y P_R\right) q_d + \text{h.c.}\right]$$
$$+ \frac{\sqrt{2}g}{2}\left[\bar{B}\not{W}\left(\kappa_L^B P_L + \kappa_R^B P_R\right) q_u + \text{h.c.}\right]$$
$$+ \frac{\sqrt{2}g}{2}\left[\bar{T}\not{W}\left(\kappa_L^T P_L + \kappa_R^T P_R\right) q_d + \text{h.c.}\right]$$
$$+ \frac{\sqrt{2}g}{2}\left[\bar{X}\not{W}\left(\kappa_L^X P_L + \kappa_R^X P_R\right) q_u + \text{h.c.}\right],$$

VLQ_v4_UFO

ζ: coupling with SM family (0 or 1)
ξ: branchings
Γ^0: kinematic functions for zero quark mass ($m_q = 0$)

$$\left(\kappa_{H,i}^Q\right)_{L/R} = \kappa_Q \frac{m_Q}{v} \sqrt{\frac{\zeta_{L/R}^i \xi_H^Q}{\Gamma_H^0}}, \qquad \left(\kappa_{W,i}^Q\right)_{L/R} = \kappa_Q \sqrt{\frac{\zeta_{L/R}^i \xi_W^Q}{\Gamma_W^0}}, \qquad \left(\kappa_{Z,i}^Q\right)_{L/R} = \kappa_Q \sqrt{\frac{\zeta_{L/R}^i \xi_Z^Q}{\Gamma_Z^0}}$$

The VLQcalc module only takes into account interactions with the 3rd family SM quarks, as their couplings are more dominant compared to other families ($i = 3$).

$$\kappa_{H,L/R}^Q = \kappa_Q \frac{m_Q}{v} \sqrt{\frac{\zeta_{L/R} \xi_H^Q}{\Gamma_H^0}}, \qquad \kappa_{W,L/R}^Q = \kappa_Q \sqrt{\frac{\zeta_{L/R} \xi_W^Q}{\Gamma_W^0}}, \qquad \kappa_{Z,L/R}^Q = \kappa_Q \sqrt{\frac{\zeta_{L/R} \xi_Z^Q}{\Gamma_Z^0}}$$

$$\bar{B}\not{W}\left(\kappa_L^B P_L + \kappa_R^B P_R\right) q_u = \kappa[\bar{B}_{L/R} W_\mu^- \gamma^\mu u_{L/R}^i]$$
$$\bar{B}\not{Z}\left(\tilde{\kappa}_L^B P_L + \tilde{\kappa}_R^B P_R\right) q_d = \tilde{\kappa}[\bar{B}_{L/R} Z_\mu \gamma^\mu d_{L/R}^i]$$
$$h\bar{B}\left(\hat{\kappa}_L^B P_L + \tilde{\kappa}_R^B P_R\right) q_d = \hat{\kappa}[\bar{B}_{L/R} H d_{L/R}^i]$$

*Q and q denotes respectively vector-like quarks and 3rd family SM quarks

Main Features: Converting couplings

```python
import VLQcalc.model as model

vlq = model.VLQ(VLQ_type='B', FNS=4, LR=False)
Mass = [1000, 1500, 2000]
vlq.setMass(Mass)

###################################
kB = 0.5
branchings = [0.25, 0.5, 0.25] # [BR(Hq), BR(Wq), BR(Zq)]
vlq.convertModel(kB, branchings, reverse=False)  # converts kappa from VLQ_UFO to VLQ_v4_UFO

print('VLQ_v4_UFO:')
print('mB [GeV]\tkH\tkW\tkZ')
for i in range(len(Mass)):
    M = Mass[i]
    kH = round(vlq.KappaH[i], 2)
    kW = round(vlq.KappaW[i], 2)
    kZ = round(vlq.KappaZ[i], 2)
    print(f'{M}\t\t{kH}\t{kW}\t{kZ}')

###################################
newKappas = [vlq.KappaH, vlq.KappaW, vlq.KappaZ]
vlq.convertModel(newKappas, branchings, reverse=True)  # converts kappas from VLQ_v4_UFO to VLQ_UFO

print('VLQ_UFO:')
print('mB [GeV]\tkB')
for i in range(len(Mass)):
    M = Mass[i]
    kB = round(vlq.KappaOld[i], 2)
    print(f'{M}\t\t{kB}')
```

Output:

VLQ_v4_UFO:

mB [GeV]	kH	kW	kZ
1000	1.45	0.35	0.35
1500	2.16	0.35	0.35
2000	2.87	0.35	0.35

Warning !
Deleting old kappa values for kB

VLQ_UFO:

mB [GeV]	kB
1000	0.5
1500	0.5
2000	0.5

Main Features: Converting couplings

```python
import VLQcalc.model as model

vlq = model.VLQ('Y', FNS=5, LR=True)
Mass = [1000, 1500, 2000]
vlq.setMass(Mass)

###################################
kY = 0.5
vlq.convertModel(kY)  # converts kappa from VLQ_UFO to VLQ_v4_UFO

print('VLQ_v4_UFO:')
print('mY [GeV]\tkW')
for i in range(len(Mass)):
    M = Mass[i]
    kW = round(vlq.KappaW[i], 2)
    print(f'{M}\t\t{kW}')

###################################
newKappas = vlq.KappaW
vlq.convertModel(newKappas, reverse=True)  # converts kappas from VLQ_v4_UFO to VLQ_UFO

print('VLQ_UFO:')
print('mY [GeV]\tkY')
for i in range(len(Mass)):
    M = Mass[i]
    kY = round(vlq.KappaOld[i], 2)
    print(f'{M}\t\t{kY}')
```

Output:

```
VLQ_v4_UFO:
mY [GeV]        kW
   1000         0.5
   1500         0.5
   2000         0.5

Warning !
Deleting old kappa values for kY

VLQ_UFO:
mY [GeV]        kY
   1000         0.5
   1500         0.5
   2000         0.5
```

Main Features: Decay width and branching ratio calculations

VLQ_v4_UFO model:

```
import VLQcalc.model as model

vlq = model.VLQ('T')
kappas = [1.612, 0.384, 0.402]

print('\nmT [GeV]\tGamma [GeV]\tBR(Ht)\tBR(Wb)\tBR(Zt)')
for mass in range(1000,2001,200):
    decayH, decayW, decayZ, Gamma = vlq.calcDecay(mass, kappas)
    BRH = round(decayH/Gamma, 3)
    BRW = round(decayW/Gamma, 3)
    BRZ = round(decayZ/Gamma, 3)
    Gamma = round(Gamma, 3)
    print(f'{mass}\t\t{Gamma}\t\t{BRH}\t{BRW}\t{BRZ}')
```

Output:

mT [GeV]	Gamma [GeV]	BR(Ht)	BR(Wb)	BR(Zt)
1000	100.011	0.25	0.5	0.25
1200	161.189	0.188	0.536	0.276
1400	244.618	0.146	0.561	0.294
1600	354.031	0.115	0.579	0.306
1800	493.154	0.093	0.591	0.315
2000	665.709	0.077	0.601	0.322

```
import VLQcalc.model as model

vlq = model.VLQ('T', LR=True)
kappas = [0.984, 0.272, 0.285]

print('\nmT [GeV]\tGamma [GeV]\tBR(Ht)\tBR(Wb)\tBR(Zt)')
for mass in range(1000,2001,200):
    decayH, decayW, decayZ, Gamma = vlq.calcDecay(mass, kappas, vlq.LR)
    BRH = round(decayH/Gamma, 3)
    BRW = round(decayW/Gamma, 3)
    BRZ = round(decayZ/Gamma, 3)
    Gamma = round(Gamma, 3)
    print(f'  {mass}\t\t{Gamma}\t\t{BRH}\t{BRW}\t{BRZ}')
```

Output:

mT [GeV]	Gamma [GeV]	BR(Ht)	BR(Wb)	BR(Zt)
1000	100.062	0.25	0.501	0.249
1200	160.205	0.181	0.541	0.278
1400	242.665	0.136	0.567	0.296
1600	351.204	0.105	0.585	0.309
1800	489.567	0.084	0.598	0.319
2000	661.495	0.068	0.607	0.325

Main Features: Decay width and branching ratio calculations

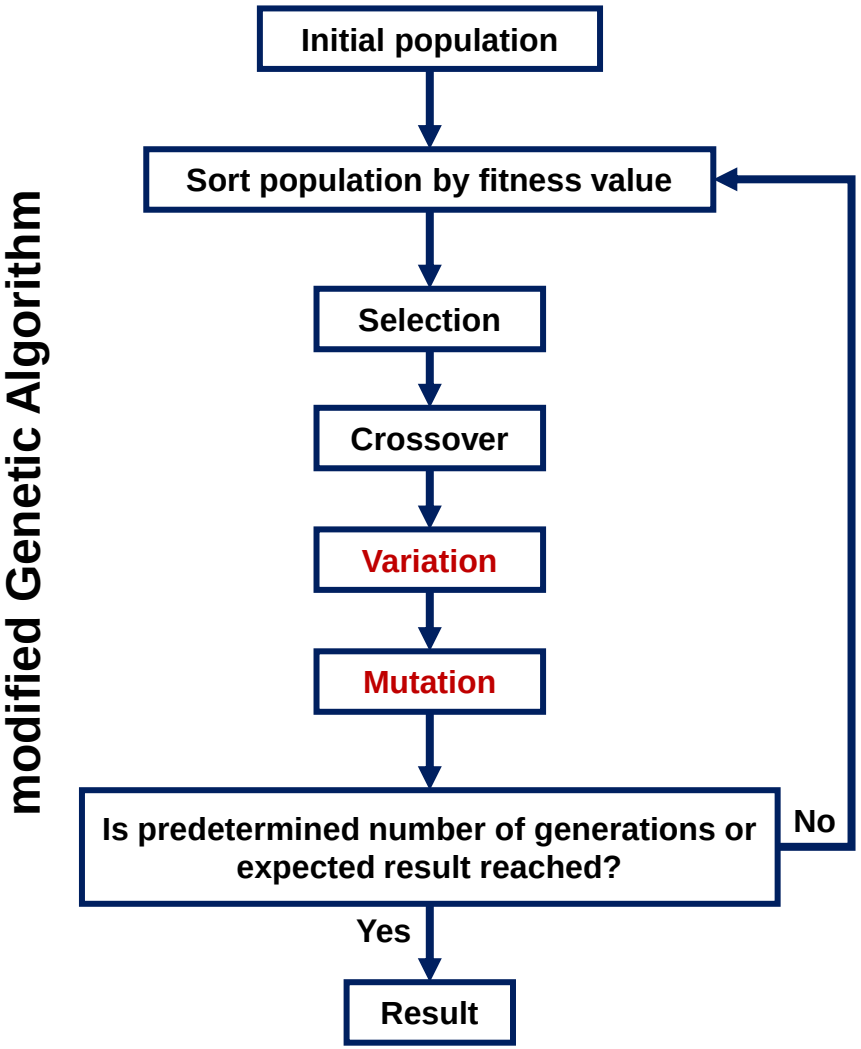
VLQ_UFO model:

```
import VLQcalc.model as model

Mass = range(1000, 2001, 200)
branchings = [0.25, 0.5, 0.25] # [BR(Hb), BR(Wt), BR(Zb)]
vlq = model.VLQ('T')
vlq.setMass( Mass )

kT = 0.5
vlq.convertModel(kT, branchings)  # converts kappa from VLQ_UFO to VLQ_v4_UFO

kappas = []
for i in range(len(Mass)):
    kappas.append( [vlq.KappaH[i], vlq.KappaW[i], vlq.KappaZ[i]] )

print('\nmT [GeV]\tGamma [GeV]\tBR(Ht)\tBR(Wb)\tBR(Zt)')
for i in range(len(Mass)):
    decayH, decayW, decayZ, Gamma = vlq.calcDecay(Mass[i], kappas[i])
    BRH = round(decayH/Gamma, 3)
    BRW = round(decayW/Gamma, 3)
    BRZ = round(decayZ/Gamma, 3)
    Gamma = round(Gamma, 3)
    print(f'{Mass[i]}\t\t{Gamma}\t\t{BRH}\t{BRW}\t{BRZ}')
```

Output:

mT [GeV]	Gamma [GeV]	BR(Ht)	BR(Wb)	BR(Zt)
1000	82.032	0.247	0.517	0.236
1200	142.742	0.246	0.513	0.241
1400	227.621	0.245	0.511	0.244
1600	340.697	0.244	0.51	0.246
1800	485.999	0.244	0.509	0.247
2000	667.553	0.244	0.508	0.248

Main Features: Optimization of couplings for Γ/m ratio



Various physics analyses, both theoretical and experimental, are carried out in the models to determine the precise values of the independent parameters κ, ζ, ξ, and m_Q. Some of these analyses utilize the Narrow Width Approximation (NWA) to simplify the complexity in particle decays. NWA is grounded on the premise that the decay width of a specific particle is significantly smaller than its mass. In this context, the decay of the particle happens almost instantaneously compared to other potential interactions, with no additional interactions occurring during the decay process. This simplification helps reduce the number of independent parameters and streamlines the analyses. The VLQcalc module employed a **modified Genetic Algorithm** [5] (**modifiedGA**) to calculate coupling constants within the Standard Model's 3rd family that are compatible with the NWA.

Detailed information and performance tests are on the additional slides and at GitHub page.

Main Features: Optimization of couplings for Γ/m ratio

VLQ_v4_UFO model:

```
import VLQcalc.model as model

Mass = range(1000, 2001, 200)
branchings = [0.25, 0.5, 0.25] # [BR(Hb), BR(Wt), BR(Zb)]
GM = 0.1 # Gamma/Mass ratio

vlq = model.VLQ('T')
vlq.setMass(Mass)
vlq.calcRatioKappas(branchings, GM)

print('\nmT [GeV]\tkH\tkW\tkZ')
for i in range(len(Mass)):
    M = Mass[i]
    kH = round(vlq.KappaH[i], 3)
    kW = round(vlq.KappaW[i], 3)
    kZ = round(vlq.KappaZ[i], 3)
    print(f'{M}\t\t{kH}\t{kW}\t{kZ}')
```

Output:

mT [GeV]	kH	kW	kZ
1000	1.612	0.384	0.402
1200	1.603	0.32	0.33
1400	1.598	0.274	0.281
1600	1.595	0.24	0.244
1800	1.593	0.213	0.216
2000	1.592	0.192	0.194

```
print('\nmT [GeV]\tGamma [GeV]\tBR(Hb)\tBR(Wt)\tBR(Zb)')
for i in range(len(Mass)):
    kappas = [vlq.KappaH[i], vlq.KappaW[i], vlq.KappaZ[i]]
    decayH, decayW, decayZ, Gamma = vlq.calcDecay( Mass[i], kappas )
    BRH = round(decayH/Gamma, 2)
    BRW = round(decayW/Gamma, 2)
    BRZ = round(decayZ/Gamma, 2)
    Gamma = round(Gamma, 2)
    print(f'{Mass[i]}\t\t{Gamma}\t\t{BRH}\t{BRW}\t{BRZ}')
```

Output:

mT [GeV]	Gamma [GeV]	BR(Hb)	BR(Wt)	BR(Zb)
1000	100.0	0.25	0.5	0.25
1200	120.0	0.25	0.5	0.25
1400	140.0	0.25	0.5	0.25
1600	160.0	0.25	0.5	0.25
1800	180.0	0.25	0.5	0.25
2000	200.0	0.25	0.5	0.25

Main Features: Optimization of couplings for Γ/m ratio

VLQ_UFO model:

```
import VLQcalc.model as model

Mass = range(1000, 2001, 200)
branchings = [0.25, 0.5, 0.25] # [BR(Hb), BR(Wt), BR(Zb)]
GM = 0.1 # Gamma/Mass ratio

vlq = model.VLQ('T')
vlq.setMass(Mass)
vlq.calcRatioKappas(branchings, GM)

newKappas = [vlq.KappaH, vlq.KappaW, vlq.KappaZ]
vlq.convertModel(newKappas,branchings,reverse=True)  # converts kappas VLQ_v4_UFO to VLQ_UFO

print('mB [GeV]\tkB')
for i in range(len(Mass)):
    M = Mass[i]
    kB = round(vlq.KappaOld[i], 2)
    print(f'{M}\t\t{kB}')
```

Output:

mB [GeV]	kB
1000	0.56
1200	0.46
1400	0.39
1600	0.34
1800	0.31
2000	0.27

Main Features: Creating MG5 input card

```
import VLQcalc.model as model
import VLQcalc.madgraph as madgraph

vlq = model.VLQ('B')
vlq.setMass( range(1000, 2001, 200) )
vlq.BRs = [0.25, 0.5, 0.25]
vlq.KappaOld = [0.5]

mg5 = madgraph.MG5(vlq, 'VLQ_UFO')

mg5.setProcess( 'p p > bp j, (bp > w- t, t > w+ b)' )

mg5.shower = 'Pythia8'
mg5.detector = 'Delphes'

mg5.addInput('Nevents 1000') # inputs for cards

mg5.createMG5Input('mg5_input') # name of input file
```

When creating an input card, the processes defined in *setProcess* and *addProcess* are expanded to include both particles and antiparticles. This ensures the most accurate event production statistically.

mg5_input.dat:

```
import model VLQ_UFO
define VLB = bp bp~
define WW = w+ w-
define tt = t t~
define bb = b b~
generate p p > VLB j, (VLB > WW tt, tt > WW bb)
launch
shower=Pythia8
detector=Delphes
analysis=OFF
madspin=OFF
reweight=OFF
done
set Nevents 1000
set zetaBdL 0.000000e-01
set zetaBdR 0.000000e-01
set zetaBsL 0.000000e-01
set zetaBsR 0.000000e-01
set zetaBbL 1.000000e+00
set zetaBbR 0.000000e-01
set xibph 0.25
set xibpw 0.5
set xibpz 0.25
set MBP scan1:[1000, 1200, 1400, 1600, 1800, 2000]
set KB 0.5
done
```

Main Features: Creating MG5 input card

```
import VLQcalc.model as model
import VLQcalc.madgraph as madgraph

vlq = model.VLQ('B', LR=True)
vlq.setMass( range(1000, 2001, 200) )
vlq.calcRatioKappas([0.25, 0.5, 0.25], 0.1)

mg5 = madgraph.MG5(vlq, 'VLQ_v4_UFO')

mg5.setProcess( 'p p > bp j, (bp > w- t, t > w+ b)' )
mg5.addProcess( 'p p > bp j b, (bp > w- t, t > w+ b)'
)

mg5.shower = 'Pythia8'
mg5.detector = 'Delphes'

mg5.addInput('Nevents 1000') # inputs for cards

mg5.createMG5Input('mg5_input') # name of input file
```

mg5_input.dat:

```
import model VLQ_v4_UFO_4FNS_UFO-3rd
define VLB = bp bp~
define WW = w+ w-
define tt = t t~
define bb = b b~
generate p p > VLB j, (VLB > WW tt, tt > WW bb)
add process p p > VLB j bb, (VLB > WW tt, tt > WW bb)
launch
shower=Pythia8
detector=Delphes
analysis=OFF
madspin=OFF
reweight=OFF
done
set Nevents 1000
set MBP scan1:[1000, 1200, 1400, 1600, 1800, 2000]
set KBLh3 scan1:[1.134016, 1.129345, 1.126635, 1.12494, 1.12382, 1.123047]
set KBRh3 scan1:[1.134016, 1.129345, 1.126635, 1.12494, 1.12382, 1.123047]
set KBLw3 scan1:[0.285178, 0.233964, 0.198706, 0.172851, 0.153037, 0.137346]
set KBRw3 scan1:[0.285178, 0.233964, 0.198706, 0.172851, 0.153037, 0.137346]
set KBLz3 scan1:[0.271565, 0.226281, 0.193945, 0.169698, 0.15084, 0.135755]
set KBRz3 scan1:[0.271565, 0.226281, 0.193945, 0.169698, 0.15084, 0.135755]
done
```

Results and Future Plans

VLQcalc,

- allows for the conversion of couplings between existing model files,
- calculates decay widths and branching ratios based on the given masses and couplings,
- optimizes couplings based on the Γ_Q/m_Q ratio and branching ratios,
- generates MG5 input files for the given processes.

These features accelerate the event generation stages in physics studies.

Future plans:

- The FeynRules model page will be regularly monitored. If a new VLQ model is added, it will be integrated into the module.
- New features can be added based on recommendations.

References

1. N. D. Christensen and C. Duhr. Feynrules–Feynman rules made easy. *Computer Physics Communications*,180(9):1614–1641, 2009.
2. M. Buchkremer, G. Cacciapaglia, A. Deandrea, and L. Panizzi. Model-independent framework for searches of top partners. *Nuclear Physics B*, 876(2):376–417, 2013.
3. B. Fuks and H. S. Shao. QCD next-to-leading-order predictions matched to parton showers for vector-like quark models. *The European Physical Journal C*, 77(2):1–21, 2017.
4. J. Alwall, M. Herquet, F. Maltoni, O. Mattelaer, and T. Stelzer. Madgraph 5: going beyond. *Journal of High Energy Physics*, 2011(6):1–40, 2011.
5. A. C. Canbay. modifiedGA (Version 0.1.0) [Computer software]. DOI: 10.5281/zenodo.12569505, GitHub: acanbay/modifiedGA

Additional Slides: VLQ Object

```
import VLQcalc.model as model


vlq = model.VLQ(VLQ_type, FNS, LR)
```

Parameter	Format	Values	Default Value
VLQ_type	string	X, T, B, Y	-
FNS	integer	4, 5	4
LR	bool	True, False	False

- $m_b = 0$ when FNS=5
- LR=False allows calculations with only left-handed couplings, while LR=True allows calculations with both left and right-handed couplings.

```
vlq.setMass( 1000 )
vlq.setMass( [1000, 1500, 2000] )
vlq.setMass( range(1000, 2001, 200) )
```

Additional Slides: `convertModel`

```
vlq.convertModel(Kappas, BRs, reverse)
```

Parameter	Format	Values	Default Value
`Kappas`	float/integer or list	any	-
`BRs`	float/integer or list	any	1
`reverse`	bool	`True, False`	`False`

- `Kappas` is
  - κ_Q where Q is X, T, B or Y for converting couplings from VLQ_UFO to VLQ_v4_UFO.
  - given in the list as $[\kappa_H, \kappa_W, \kappa_Z]$ for T and B, while it is only κ_W for X and Y when converting couplings from VLQ_v4_UFO to VLQ_UFO.
- For T and B, the parameter named `BRs` is given in the list as [BR($Q \to Hq$), BR($Q \to Wq$), BR($Q \to Zq$)] where q represents the 3rd family quarks of the Standard Model. Only BR($Q \to Wq$) is specified for X and Y.
- When the `reverse` value is `False`, the conversion is from VLQ_UFO to VLQ_v4_UFO; when `True`, it is from VLQ_v4_UFO to VLQ_UFO.

Additional Slides: `calcDecay`

It only works with the **VLQ_v4_UFO** model. Converted couplings can be used for **VLQ_UFO** model

```
decayH, decayW, decayZ, Gamma = vlq.calcDecay(Mass, Kappas, LR)
```

for T and B

```
decayW = vlq.calcDecay(Mass, Kappa, LR)
```

for X and Y

Parameter	Format	Values	Default Value
`Mass`	float/integer or list	any	-
`Kappas`	list	any	-
`Kappa`	float/integer	any	-
`LR`	bool	`True, False`	`False`

Additional Slides: `calcRatioKappas`

It is used to calculate the couplings according to the Γ/m ratio. It only works with the VLQ_v4_UFO model. Couplings can be converted to VLQ_UFO model after calculations are made according to this model.

modifiedGA is used to calculate couplings according to the ratio.

```
vlq.calcRatioKappas(BRs, Ratio)
```

`Ratio` is Γ_Q/m_Q value

Additional Slides: MG5 Object

```
import VLQcalc.madgraph as madgraph


mg5 = madgraph.MG5(VLQ_object, Model)
```

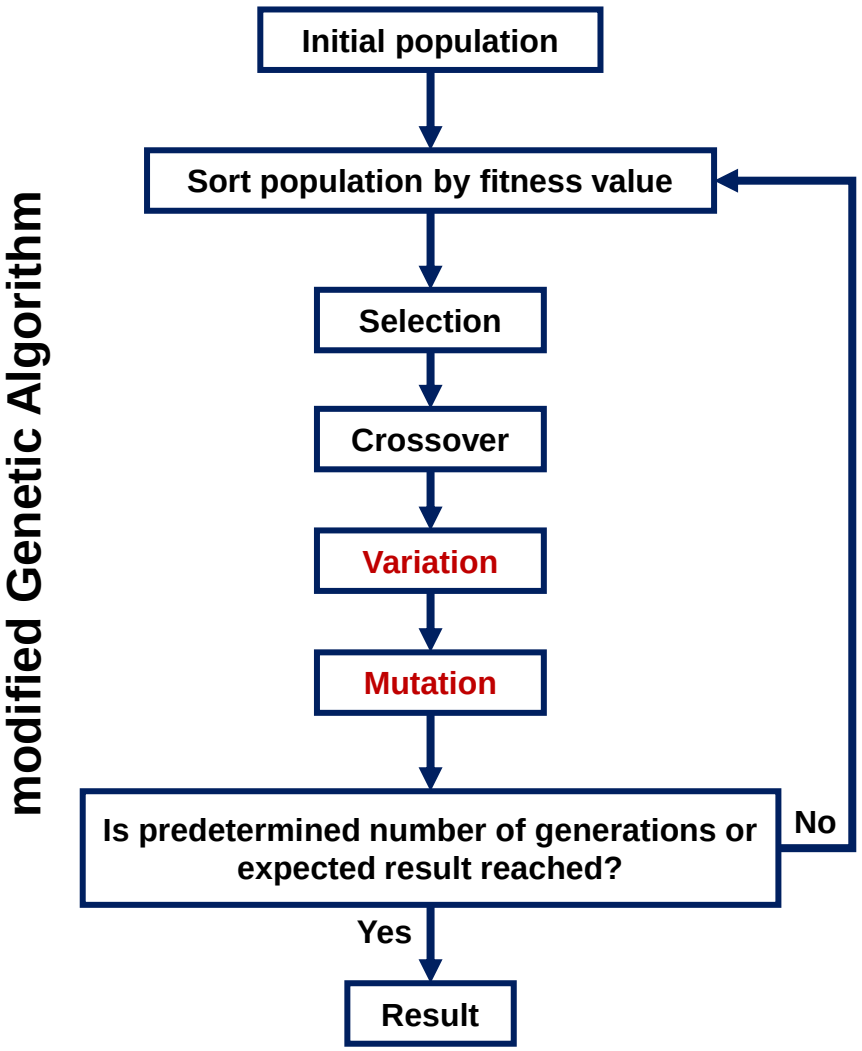
model parameter can be VLQ_UFO or VLQ_v4_UFO in string format.

```
mg5.setProcess(process)
mg5.addProcess(process)
mg5.addInput(input)
mg5.createMG5Input(file_name)
```

string format

Property	Values	Default Value
shower	OFF, pythia8, ...	OFF
detector	OFF, Delphes, ...	OFF
analysis	OFF, ExRoot, MadAnalysis, ...	OFF
madspin	OFF, ON, onshell, full	OFF
reweight	OFF, ON	OFF

Additional Slides: modifiedGA



In order to address challenges related to multiple and correlated parameters, it is essential to maintain a large population and generation count in the Genetic Algorithm (GA). However, this can lead to longer computation times.

To tackle this issue, we introduced the concept of variation to the genetic algorithm, drawing inspiration from nature itself. We modified the mutation distribution to follow a Gaussian distribution, with the standard deviation adjusted based on the results of each generation.

What is variation?

The best genes undergo minor changes as they are passed on to the next generation

modifiedGA GitHub page

Additional Slides: modifiedGA

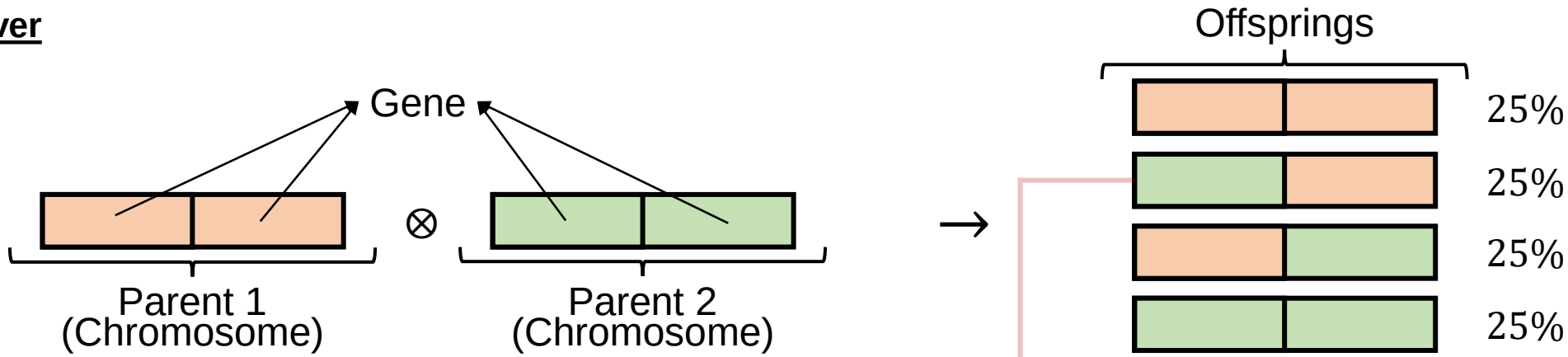
- **Initial population** consists of numbers determined to be within the domain of the variables.
- The initial population is tested whether the relevant function gives the optimized value and the **fitness** values (closeness to the result) are determined.
- The initial population is weighted according to their fitness values, and some of the highest weighted values are **selected** for the next generation.
- The parents of the offsprings which will form the next generation are selected by **weighted wheel (roulette wheel) selection** according to their fitness values.

weightless wheel

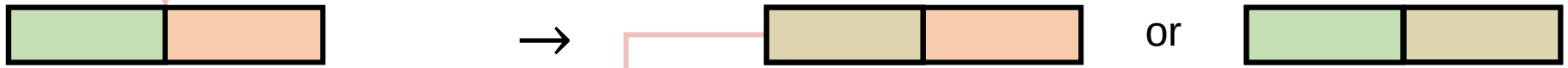
weighted wheel

Additional Slides: modifiedGA

Crossover



Variation



The variation is based on the genes of the parents.
The new gene is the average of the parent genes.

Mutation

In mutation, already existing genes are changed
within specified deviation probabilities.

Additional Slides: modifiedGA

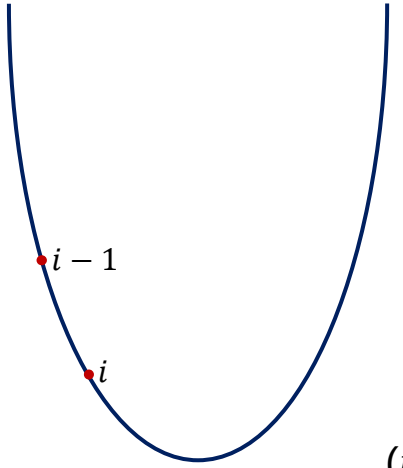
Standard Mutation:

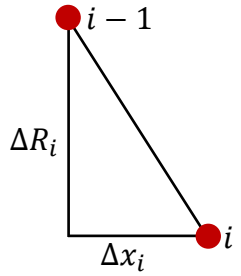
$$x_i = x_i \times random\boldsymbol{Uniform}(0.9, 1.1)$$

(i: Number of generations) These ratios may vary depending on usage

Modified Mutation:

$$x_i = x_i \times random\boldsymbol{Gauss}(\mu = 1, \sigma)$$





(i: Number of generations)

$$\Delta R_i = Result_i - Result_{i-1}$$

$$\Delta x_i = \sqrt{\sum_{k=1}^{N} \left(x_{k,i} - x_{k,i-1}\right)^2}$$

(N: Number of parameters)

$$\sigma_i = \frac{\Delta R_i}{\Delta x_i}$$

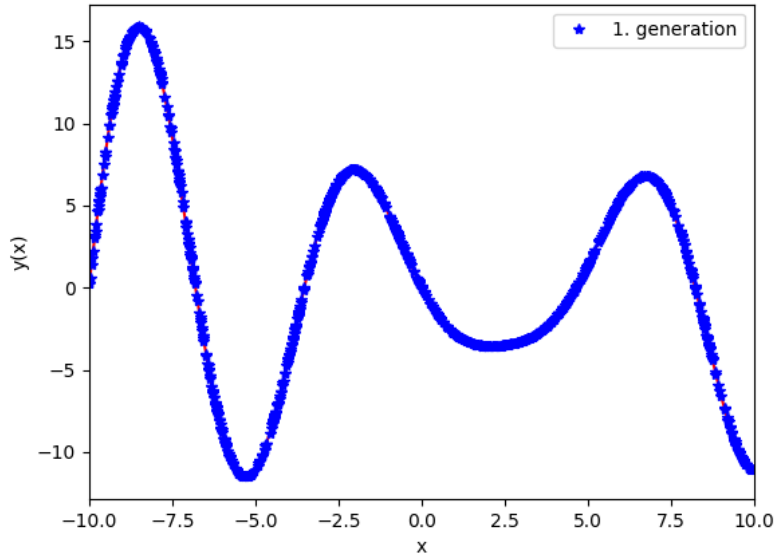
We want a fluctuation at most σ around the parameter for the mutation since variation is already applied.

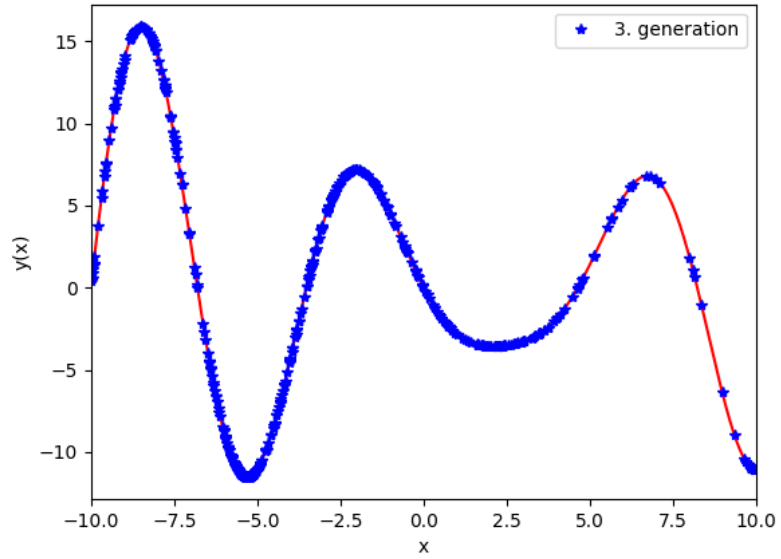
Additional Slides: modifiedGA

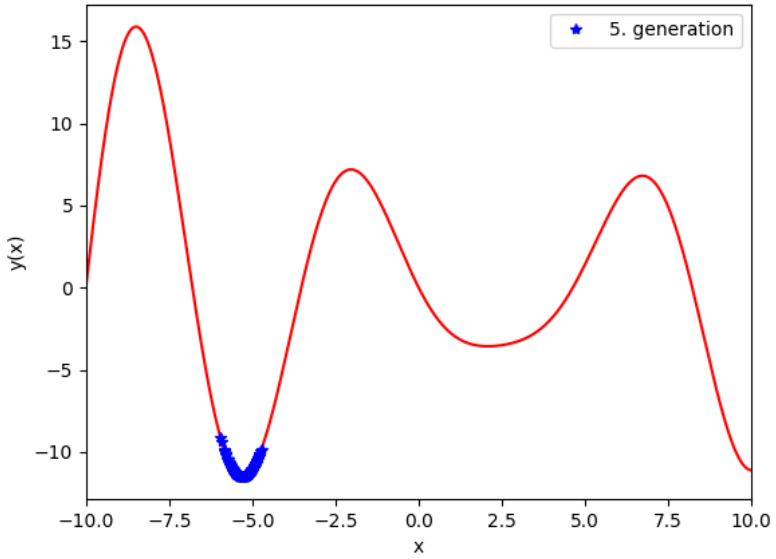
Minimize

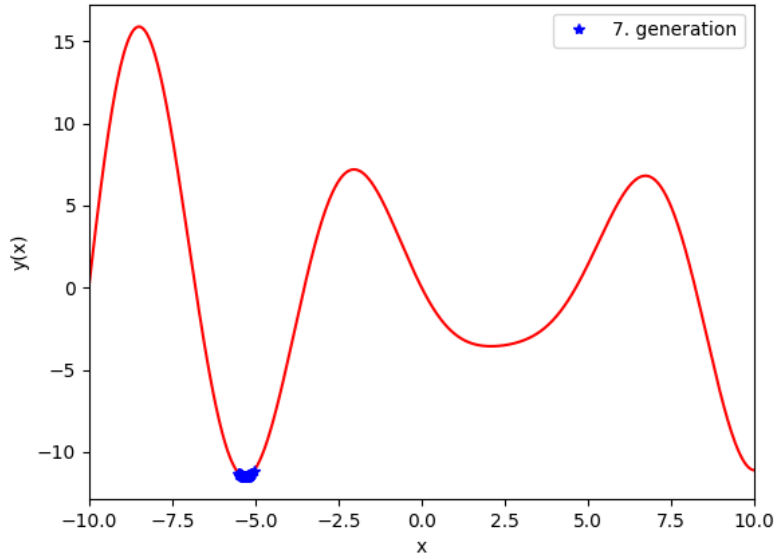
$f(x) = sin(x)(x-5) + cos(x)x$

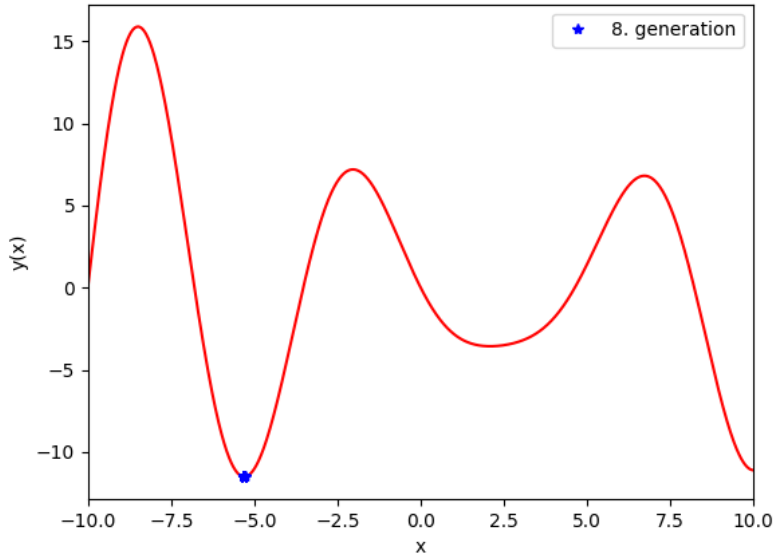
$x: [-10,10]$

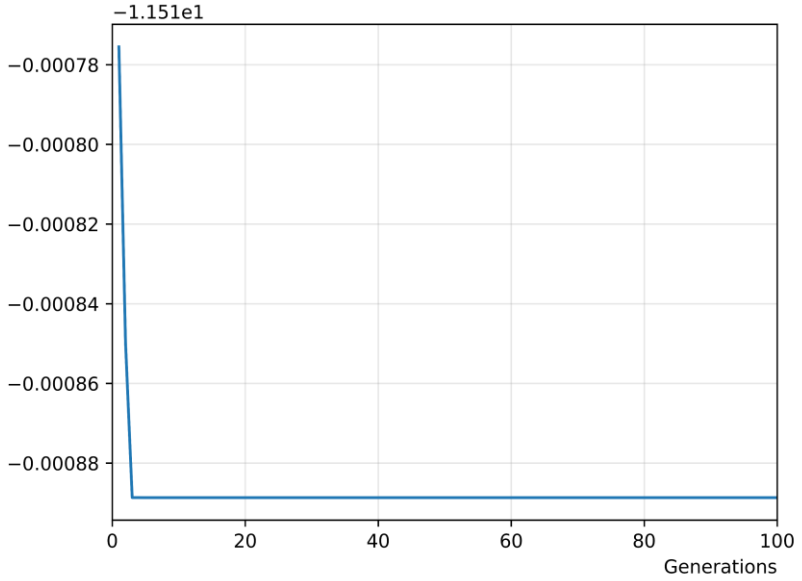












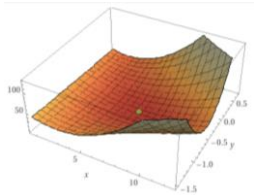
Additional Slides: modifiedGA

minimize $(x-5)^2+(y-3)^2+x^2y^2+x^2y$

Global minimum

$\min\{(x-5)^2+(y-3)^2+x^2y^2+x^2y\} \approx 3.64497208987637341623632593 7$
at $(x, y) \approx (6.612677980974700535797561234, -0.421748382735042090930087160 4)$

WolframAlpha computational intelligence.



pymoo: Multi-objective Optimization in Python

pymoo is a multi-objective optimization module for python that contains many algorithms.

In order to test the algorithms under equal conditions, the following definitions were made on pymoo:

- algorithm = "ga" (Genetic Alrgorithm)
- N_offsprings = None
- get_sampling("real_random")
- crossover = get_crossover("real_sbx")
- mutation = get_mutation("real_pm")
- eliminate_duplicates = False

Population Size : 100
Number of Generations : 100
Distribution Size : 1000

		Domain		
		[-10,10]	**[-100,100]**	**[-1000,1000]**
pymoo (0.4.0)	**σ (around the correct result)**	2.492×10^{-5}	3.091×10^{-3}	1.823×10^{0}
	Δt [m:s]	10:17	09:39	09:51

		Domain		
		[-10,10]	**[-100,100]**	**[-1000,1000]**
modified	**σ (around the correct result)**	5.108×10^{-15}	5.225×10^{-15}	5.159×10^{-15}
	Δt [m:s]	01:02	01:05	01:05

This time difference is due to the time spent on importing modules, random number generations and evaluation functions. This time difference is expected to be approximately the same for different problems (with not correlated variables).

The results in both algorithms may vary according to eta and probability values. Therefore, it is possible to improve both algorithms. Here the default values are compared.