OpenSRANE

Open Software for Risk Assessment of NaTech Events

by

Bijan Sayyafzadeh

As final thesis project for the degree of

Doctor of Philosophy

In

Engineering – Structure Engineering

Guide masters:

* Dr. Mahdi Sharifi
* Dr. Abdolreza Sarvghad Moghaddam
* Dr. Eslam Kashi

Table of Contents

[Introduction 4](#_Toc132566314)

[Verification 5](#_Toc132566315)

[Hazard 5](#_Toc132566316)

[Earthquake Module 5](#_Toc132566317)

[DateAndTime 6](#_Toc132566318)

[DateTime Module 6](#_Toc132566319)

[WindData 7](#_Toc132566320)

[WindRose Module 7](#_Toc132566321)

[Sites 9](#_Toc132566322)

[Site Module 9](#_Toc132566323)

[Substance 10](#_Toc132566324)

[Material Module 10](#_Toc132566325)

[DataBank Module 10](#_Toc132566326)

[Fragilities 10](#_Toc132566327)

[Fragility 10](#_Toc132566328)

[Probit Module 12](#_Toc132566329)

[OutFlowModels 14](#_Toc132566330)

[GasUnitHole Module 14](#_Toc132566331)

[TankHole Family 16](#_Toc132566332)

[Simultaneous Family 19](#_Toc132566333)

[NoOutFlow Module 20](#_Toc132566334)

[Connectors 20](#_Toc132566335)

[DS\_LOC module 20](#_Toc132566336)

[Pb\_LOC module 21](#_Toc132566337)

[Out\_Physic module 21](#_Toc132566338)

[DispersionSpreadModels 21](#_Toc132566339)

[LiquidSpread Module 21](#_Toc132566340)

[BritterMcQuaid Module 23](#_Toc132566341)

[GasGuassian Module 24](#_Toc132566342)

[LqdSprdGuassianGasDisp Module 24](#_Toc132566343)

[PhysicalEffect SubPackage 30](#_Toc132566344)

[fire\_point\_source module 30](#_Toc132566345)

[VCE\_TNT module 32](#_Toc132566346)

[Safe module 33](#_Toc132566347)

# Introduction

In this document, for each module, using an example or examples, the module performance has been investigated and confirmed or verified. Examples are provided form reliable references or from solved defined examples in reliable software (like excel) and in this way we get ensure of the modules results.

Also, to check the performance of the modules, some OpenSRANE python scripts has been provided to check the module results. Provided scripts in some cases are a little different from the usual OpenSRANE commands because need of getting access to some special parameters.

This is obvious that modules that provided by developers should also have verifications in this document and for this purpose, this document is presented in open access format.

# Verification

## Hazard

### Earthquake Module

In this module there is no special algorithm. Only a list of the probabilities and magnitudes should be entered by the user. An example has been presented in the user’s guideline file. Using a python random number generating module, a random number between 0 and1 will be generated and the module returns its corresponding magnitude. To ensure the generated files are compatible with the entered values, the results are checked using MonteCarlo approach as the following:

opr**.**wipe**()**

#User Hazard Definition

PGA**=[**1.4**,** 1.29984**,**1.27091**,**1.24865**,**1.22194**,**1.20191**,**1.17297**,**1.13959**,**1.11065**,**1.08172**,**

1.05501**,**1.03275**,**0.994913**,**0.961526**,**0.92814**,**0.899205**,**0.872496**,**0.843561**,**0.816852**,**

0.790143**,**0.761208**,**0.732273**,**0.696661**,**0.667727**,**0.636566**,**0.609857**,**0.589825**,**0.569793**,**

0.549762**,**0.525278**,**0.503021**,**0.48744**,**0.469634**,**0.451828**,**0.431797**,**0.409539**,**0.38283**,**

0.356121**,**0.338315**,**0.307154**,**0.280445**,**0.260413**,**0.23593**,**0.211447**,**0.17806**,**0.158029**,**

0.142448**,**0.126868**,**0.113514**,**0.0979332**,**0.0845787**,**0.0712242**,**0.0578696**,**0.0534181**,**

0.0489666**,**0.0445151**,**0.0356121**,**0.0356121**,**0.0311606**]**

Prob**=[**0**,** 0.000000446937**,**0.000000529832**,**0.000000607087**,**0.000000695608**,**0.000000824624**,**0.000000944864**,**

0.00000115888**,**0.00000137382**,**0.00000157414**,**0.0000019307**,**0.00000206667**,**0.00000262252**,**0.00000321654**,**

0.00000381312**,**0.00000452035**,**0.00000535876**,**0.00000657255**,**0.00000779158**,**0.00000955643**,**0.0000113289**,**

0.0000134301**,**0.0000176322**,**0.000021626**,**0.0000265244**,**0.0000325323**,**0.0000372759**,**0.0000441896**,**0.000050633**,**0.0000621017**,**0.0000711569**,**0.0000843545**,**0.0000966544**,**0.000110748**,**0.000131288**,**0.000155639**,**0.0001975**,**

0.000259294**,**0.000318026**,**0.000403563**,**0.000548171**,**0.000744597**,**0.00104642**,**0.00152148**,**0.00253478**,**0.00356225**,**0.00467682**,**0.00657255**,**0.00892769**,**0.0109499**,**0.0143759**,**0.0209024**,**0.0274425**,**0.0325323**,**0.0399011**,**0.048939**,**0.060024**,**0.0687762**,**0.0843545**]**

obj**=**opr**.**Hazard**.**Earthquake**(**1**,**'PGA'**,**PGA**,**Prob**)** #Create Hazard Object with tag=1 that is 0th Object

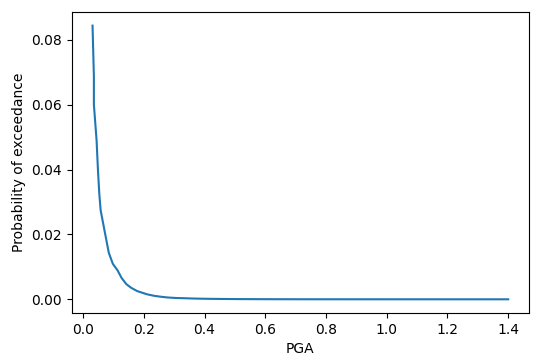


Figure - Earthquake hazard Definition

Using MonteCarlo approach, Numerous samples of random magnitude are requested from the package as the following:

#verification

Freq**=[**0 **for** i **in** PGA**]**

N**=**10000

**for** i **in** **range(**N**):**

Magnitude**=**opr**.**Hazard**.**ObjManager**.**Objlst**[**0**].**GetRandomMagnitude**()**

**if** Magnitude**==**0**:**

**continue**

**for** mag **in** PGA**:**

**if** Magnitude**>=**mag**:**

**if** Magnitude**==**mag**:**

Freq**[**PGA**.**index**(**mag**)]** **+=**1

**else:**

Freq**[**PGA**.**index**(**mag**)-**1**]** **+=**1

**break**

Freq**=[**i**/**N **for** i **in** Freq**]**

Freq**=[sum(**Freq**[**0**:**i**])** **for** i **in** **range(len(**Freq**))]**

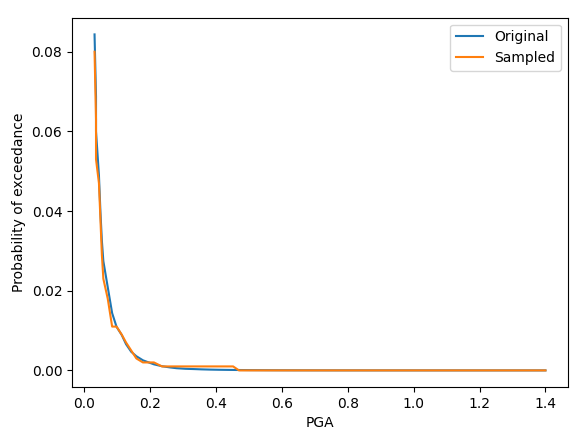


Figure - Comparison the Results of Generated Magnitudes and Original Data Entered by the User

As it is seen in in the Figure 2 the final results of the generated samples after only 1000 generation, is properly compatible on the initial user’s entered data.

## DateAndTime

### DateTime Module

In this module there is no special algorithm. According Day to night ratio defined by the user, the sampling will be done using a uniform distribution random number. To check the results of the sampling, using the following code for a day to night ratio equal to 2, Sampling has been done 1000 times and this procedure repeated for 100 times:

**import** opensrane **as** opr

**import** matplotlib**.**pyplot **as** plt

**import** random

**%**matplotlib notebook

opr**.**wipe**()**

#Generate an DateTime Obj with day to night ratio equal to 2

obj**=**opr**.**DateAndTime**.**DateTime**(**1**,**Day\_Night\_Ratio**=**2**)**

Results**=[]**

**for** i **in** **range(**100**):**

N**=**1000

Samples**=[**obj**.**isDay**()** **for** i **in** **range(**N**)]**

Results**.**append**(**Samples**.**count**(True)/**Samples**.**count**(False))**

As it is seen the results of samples are around the entered value by the user as the day to night ratio.

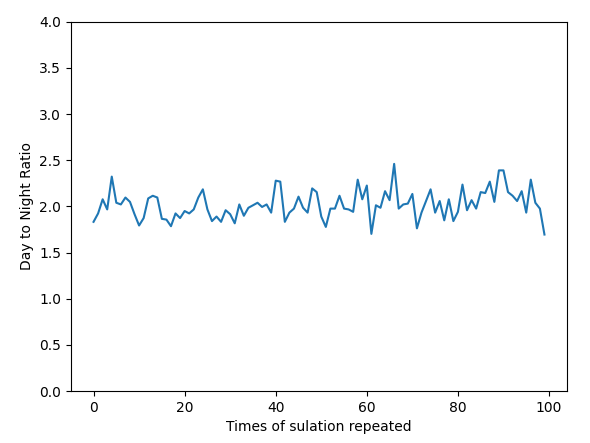


Figure - Results of day to night ratio equal 2 sampling for 1000 sampling values

## WindData

### WindRose Module

WindRose Module is responsible to return the wind sample data according user defined values. To ensure the returned values are compatible with the distribution of defined data, using a MonteCarlo approach samples are generated and checked by the initial data. At first step we have to define wind rose:

opr**.**wipe**()**

DTobj**=**opr**.**DateAndTime**.**DateTime**(**1**)**

windobj**=**opr**.**WindData**.**WindRose**(**1**)**

windobj**.**WindDayClassList**=[**'F'**,**'D'**,**'B'**,**'E'**,**'D'**,**'D'**]**

windobj**.**WindNightClassList**=[**'F'**,**'D'**,**'B'**,**'E'**,**'D'**,**'D'**]**

windobj**.**AlphaCOEFlist**=[**0.6**,**0.25**,**0.15**,**0.4**,**0.25**,**0.25**]**

windobj**.**DayWindSpeedList**=[[**1**,**2**],[**2**,**3**],[**3**,**5**],[**5**,**7**],[**7**,**9**],[**9**,**9**]]**

windobj**.**NightWindSpeedList**=[[**1**,**2**],[**2**,**3**],[**3**,**5**],[**5**,**7**],[**7**,**9**],[**9**,**9**]]**

#You don't need to define calmn Condition and program will understand it automatically

windobj**.**DayWindFreqMatrix**=[[**0.446**,**0.372**,**0.355**,**0.109**,**0.017**,**0**],**

**[**0.44**,**0.938**,**1.55**,**0.755**,**0.097**,**0.029**],**

**[**0.898**,**1.321**,**3.06**,**1.402**,**0.767**,**0.892**],**

**[**0.875**,**1.241**,**2.626**,**1.51**,**0.892**,**0.646**],**

**[**0.801**,**0.927**,**1.63**,**0.658**,**0.355**,**0.097**],**

**[**0.87**,**1.121**,**0.984**,**0.309**,**0.023**,**0.029**],**

**[**0.778**,**0.801**,**0.91**,**0.315**,**0.029**,**0**],**

**[**0.652**,**0.875**,**1.35**,**0.498**,**0.086**,**0.023**],**

**[**0.566**,**0.887**,**1.659**,**0.709**,**0.149**,**0**],**

**[**0.583**,**0.807**,**2.128**,**2.998**,**1.041**,**0.137**],**

**[**0.898**,**1.093**,**2.408**,**2.059**,**1.327**,**0.154**],**

**[**1.985**,**2.088**,**2.488**,**1.098**,**0.332**,**0.069**],**

**[**4.067**,**3.123**,**1.442**,**0.292**,**0.063**,**0.011**],**

**[**3.93**,**5.372**,**3.85**,**1.201**,**0.349**,**0.057**],**

**[**1.71**,**1.619**,**2.38**,**0.767**,**0.109**,**0.006**],**

**[**0.698**,**0.469**,**0.383**,**0.154**,**0.011**,**0**],**

**]**

#Consider night windrose equal to the day

windobj**.**NightWindFreqMatrix**=**windobj**.**DayWindFreqMatrix

The corresponding wind rose for above data is:

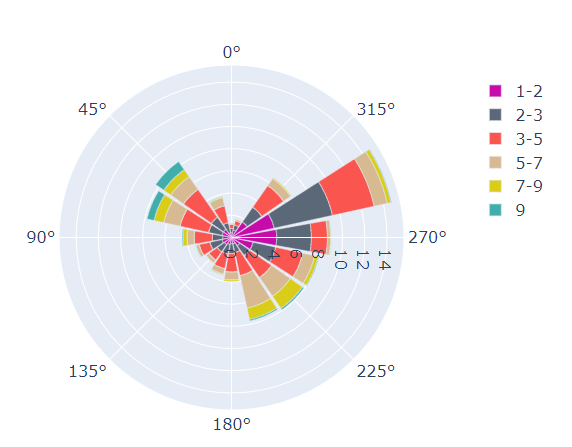


Figure - User Defined WindRose

Code to do sampling from defined windrose is written as the following:

#Verification For Day---------------------------------------

N**=**100000

windDir**=set()**

WindDirData**=dict()**

WindSpeedFreq**=[[**0.0**,**0.0**,**0.0**,**0.0**,**0.0**,**0**]** **for** i **in** **range(**16**)]**

#Sampling DayTime

DTobj**.**isDay**()**

WindDirData**[**'isCalmn'**]=**0

**for** i **in** **range(**N**):**

rslt**=**windobj**.**GetRandomWindِSample**();** #Calculate the a sample

#Calculate wind Direction----------------------------------------------

**if** rslt**[**'isCalmn'**]==False:** windDir**.**add**(**rslt**[**'WindDirection'**])**

**if** **len(**windDir**)!=len(**WindDirData**)-**1 **and** rslt**[**'isCalmn'**]==False:**

wdir**=list(**windDir**)**

add**=[**i **for** i **in** wdir **if** i **not** **in** WindDirData**.**keys**()][**0**]**

WindDirData**[**add**]=**0

**if** rslt**[**'isCalmn'**]==** **True:**

WindDirData**[**'isCalmn'**]** **=**WindDirData**[**'isCalmn'**]+**1

**else:**

WindDirData**[**rslt**[**'WindDirection'**]]** **=**WindDirData**[**rslt**[**'WindDirection'**]]+**1

#Calculating Wind Speed Frequency

**if** rslt**[**'isCalmn'**]==False:**

WindRow**=[**i **for** i **in** **range(len(**windobj**.**DayTheta**))** **if** windobj**.**DayTheta**[**i**]==**rslt**[**'WindDirection'**]][**0**]**

WindCol**=[**i **for** i **in** **range(len(**windobj**.**DayWindSpeedList**))** **if** **(**windobj**.**DayWindSpeedList**[**i**][**0**]<=**rslt**[**'WindSpeed'**]<=**windobj**.**DayWindSpeedList**[**i**][**1**])][**0**]**

WindSpeedFreq**[**WindRow**][**WindCol**]=**WindSpeedFreq**[**WindRow**][**WindCol**]+**1

#Normalize Wind MAtrix

**for** i **in** **range(len(**WindSpeedFreq**)):**

**for** j **in** **range(len(**WindSpeedFreq**[**0**])):**

WindSpeedFreq**[**i**][**j**]=**WindSpeedFreq**[**i**][**j**]/**N**\***100

**print(**'Monte Carlo Sampling Results: \n'**,** WindSpeedFreq**)**

**print(**'\n Defined Values: \n'**,**windobj**.**DayWindFreqMatrix**)**

**print()**

#Normalizing Wind Direction Data

**for** key **in** WindDirData**.**keys**():**

WindDirData**[**key**]=**WindDirData**[**key**]/**N**\***100

**print(**'\nComparision of Direction Sampling Ratio and User Defined Values:\n'**)**

userdata**=dict(zip(**windobj**.**DayTheta**,**windobj**.**DayDirectProbability**));**

**for** keys **in** userdata**.**keys**():**

**print(**f'for direction {keys} average of \t sampled values: {**round(**WindDirData**[**keys**],**3**)**} \t and UserDefined value: {**round(**userdata**[**keys**],**3**)**}'**)**

The result of the average sampled values in comparison with defined data is equal to:

for direction 0.0 average of sampled values: 1.297 and UserDefined value: 1.299

for direction 22.5 average of sampled values: 3.822 and UserDefined value: 3.809

for direction 45.0 average of sampled values: 8.329 and UserDefined value: 8.34

for direction 67.5 average of sampled values: 7.818 and UserDefined value: 7.79

for direction 90.0 average of sampled values: 4.407 and UserDefined value: 4.468

for direction 112.5 average of sampled values: 3.208 and UserDefined value: 3.336

for direction 135.0 average of sampled values: 2.82 and UserDefined value: 2.833

for direction 157.5 average of sampled values: 3.42 and UserDefined value: 3.484

for direction 180.0 average of sampled values: 3.937 and UserDefined value: 3.97

for direction 202.5 average of sampled values: 7.633 and UserDefined value: 7.694

for direction 225.0 average of sampled values: 7.997 and UserDefined value: 7.939

for direction 247.5 average of sampled values: 8.109 and UserDefined value: 8.06

for direction 270.0 average of sampled values: 9.271 and UserDefined value: 8.998

for direction 292.5 average of sampled values: 14.616 and UserDefined value: 14.759

for direction 315.0 average of sampled values: 6.703 and UserDefined value: 6.591

for direction 337.5 average of sampled values: 1.745 and UserDefined value: 1.715

Above check repeated three times and the results has been presented in the Figure 5 and as it can be seen

Figure - Comparison of sampled WindRose data with defined data

## Sites

### Site Module

This module does not produce any results and act as a storage data object. So, there is no need to any verification.

## Substance

### Material Module

This module does not produce any results and act as a storage data object. So, there is no need to any verification.

### DataBank Module

This module does not produce any results and act as a storage data object. So, there is no need to any verification.

## Fragilities

### Fragility

This module is responsible to get fragility data and return a return probability of each given magnitude. To verify available types of probability distribution (Currently normal and lognormal) the data will be checked by excel normal distribution:

opr**.**wipe**()**

obj**=**opr**.**Fragilities**.**Fragility**(**1**,**Distribution\_Type**=**'normal'**,** mean**=**5**,** StdDev**=**2**)**

#Excel results for mean=5 and std=2

Excel**=[**0.00620966532577613**,**0.00714281073527141**,**0.00819753592459613**,**0.00938670553483857**,**0.0107241100216758**,**0.0122244726550447**,**0.0139034475134986**,**0.0157776073910905**,**0.0178644205628165**,**0.0201822154057044**,**0.0227501319481792**,**0.0255880595216386**,**0.0287165598160018**,**0.0321567747956137**,**0.0359303191129258**,**0.0400591568638171**,**0.044565462758543**,**0.0494714680336481**,**0.054799291699558**,**0.060570758002059**,**0.0668072012688581**,**0.0735292596096484**,**0.0807566592337711**,**0.088507991437402**,**0.0968004845856103**,**0.105649773666855**,**0.115069670221708**,**0.12507193563715**,**0.135666060946383**,**0.146859056375896**,**0.158655253931457**,**0.171056126308482**,**0.18406012534676**,**0.197662543122692**,**0.211855398583397**,**0.226627352376868**,**0.241963652223073**,**0.257846110805865**,**0.274253117750074**,**0.291159686788346**,**0.308537538725987**,**0.32635522028792**,**0.344578258389676**,**0.363169348824381**,**0.382088577811047**,**0.401293674317076**,**0.420740290560897**,**0.440382307629757**,**0.460172162722971**,**0.480061194161628**,**0.5**,**0.519938805838372**,**0.539827837277029**,**0.559617692370243**,**0.579259709439103**,**0.598706325682924**,**0.617911422188953**,**0.636830651175619**,**0.655421741610324**,**0.67364477971208**,**0.691462461274013**,**0.708840313211654**,**0.725746882249926**,**0.742153889194135**,**0.758036347776927**,**0.773372647623132**,**0.788144601416603**,**0.802337456877308**,**0.81593987465324**,**0.828943873691518**,**0.841344746068543**,**0.853140943624104**,**0.864333939053617**,**0.87492806436285**,**0.884930329778292**,**0.894350226333145**,**0.90319951541439**,**0.911492008562598**,**0.919243340766229**,**0.926470740390352**,**0.933192798731142**,**0.939429241997941**,**0.945200708300442**,**0.950528531966352**,**0.955434537241457**,**0.959940843136183**,**0.964069680887074**,**0.967843225204386**,**0.971283440183998**,**0.974411940478361**,**0.977249868051821**,**0.979817784594296**,**0.982135579437183**,**0.98422239260891**,**0.986096552486501**,**0.987775527344955**,**0.989275889978324**,**0.990613294465161**,**0.991802464075404**,**0.992857189264729**,**0.993790334674224**]**

#Excel range of the Random Variable

Magnitude**=[**i**/**10 **for** i **in** **range(**101**)]**

#List of generated values by Created Object (obj)

Generated**=[**obj**.**GetProbability**(**i**)** **for** i **in** Magnitude**]**

plt**.**plot**(**Magnitude**,**Excel**,**label**=**'Excel'**)**

plt**.**plot**(**Magnitude**,**Generated**,**label**=**'Generated'**)**

plt**.**legend**()**

plt**.**xlabel**(**'Magnitude'**)**

plt**.**ylabel**(**'Probability'**)**

plt**.**title**(**f'CDF Distribution for distribution={obj**.**DistType} mean={obj**.**mean} and std={obj**.**StdDev}'**,**color**=**'g'**)**

plt**.**show**()**

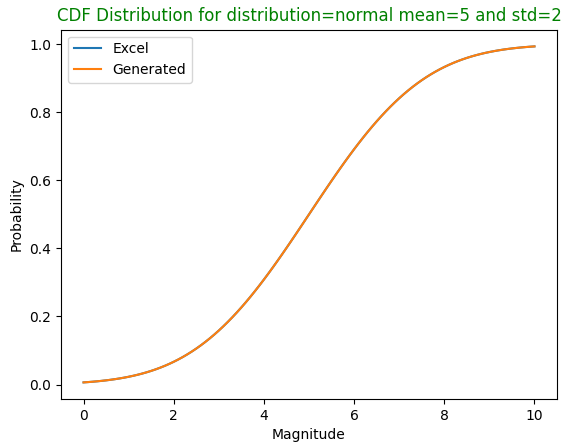


Figure - Normal Distribution Verification

And for Lognormal distribution:

opr**.**wipe**()**

**import** math

obj**=**opr**.**Fragilities**.**Fragility**(**1**,**Distribution\_Type**=**'lognormal'**,** mean**=**1.8**,** StdDev**=**0.75**)**

#Excel results for mean=51.8 and std=0.75

Excel**=[**0.000443368872962805**,**0.00819753592459613**,**0.0314866575293163**,**0.0699982906412703**,**0.119342693644532**,**0.17484684050795**,**0.232802266520937**,**0.290608642066892**,**0.346582538307854**,**0.399715801925924**,**0.449469192643485**,**0.495616760008879**,**0.538134969498259**,**0.577126148320459**,**0.612766708635067**,**0.645272716115421**,**0.67487738964635**,**0.701816715355177**,**0.72632053093662**,**0.748607266593389**,**0.768881105912544**,**0.787330726979907**,**0.804129055749149**,**0.819433649162196**,**0.833387451949063**,**0.846119757123861**,**0.85774725873404**,**0.868375125142364**,**0.878098047981419**,**0.88700123998492**,**0.895161366949546**,**0.902647407020044**,**0.909521435626017**,**0.915839337628887**,**0.921651450179559**,**0.927003140865757**,**0.931935326234779**,**0.936484935912391**,**0.940685327438584**,**0.944566656698678**,**0.948156208506431**,**0.951478691535855**,**0.954556501427705**,**0.957409955532165**,**0.960057502401805**,**0.962515908824167**,**0.96480042688445**,**0.966924943276341**,**0.968902112832907**,**0.970743478028256**,**0.972459576002877**,**0.974060034489253**,**0.975553657857638**,**0.97694850436288**,**0.978251955550021**,**0.979470778667453**,**0.980611182840105**,**0.981678869669968**,**0.98267907885608**,**0.983616629359644**]**

#Excel range of the Random Variable

Magnitude**=[**i**/**2 **for** i **in** **range(**1**,**61**)]**

#List of generated values by Created Object (obj)

Generated**=[**obj**.**GetProbability**(**i**)** **for** i **in** Magnitude**]**

plt**.**figure**()**

plt**.**plot**(**Magnitude**,**Excel**,**label**=**'Excel'**)**

plt**.**plot**(**Magnitude**,**Generated**,**label**=**'Generated'**)**

plt**.**legend**()**

plt**.**xlabel**(**'Magnitude'**)**

plt**.**ylabel**(**'Probability'**)**

plt**.**title**(**f'CDF Dist-Type={obj**.**DistType} mean of Ln(x)={obj**.**mean} and std of Ln(x)={obj**.**StdDev}'**,**color**=**'g'**)**

plt**.**show**()**

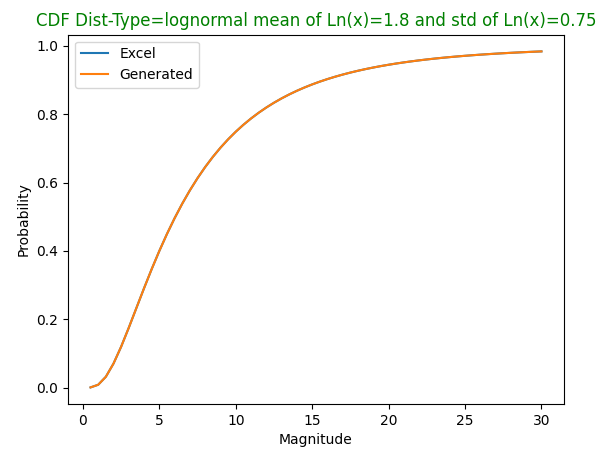


Figure - LogNormal Distribution Verification

As it is seen on Figure 6 and Figure 7 the results are fully compatible with results calculated by MS Excel.

### Probit Module

Similar to Fragilities this module is responsible to get probit data and return the corresponding probability to any entered dose. To check the performance of the module, using following probit formula, a set of data will be produce in excel and then will be check by the code:

For :

#Verification for normal distribution

opr**.**wipe**()**

obj**=**opr**.**Fragilities**.**Probit**(**1**,**Distribution\_Type**=**'normal'**,** K1**=**1**,** K2**=**0.5**)**

#Excel results for mean=5 and std=2

Excel**=[**3.39767312473005E-06**,**3.16712418331199E-05**,**0.000232629079035525**,**0.00134989803163009**,**0.00620966532577613**,**0.0227501319481792**,**0.0668072012688581**,**0.158655253931457**,**0.308537538725987**,**0.5**,**0.691462461274013**,**0.841344746068543**,**0.933192798731142**,**0.977249868051821**,**0.993790334674224**,**0.99865010196837**,**0.999767370920964**,**0.999968328758167**,**0.999996602326875**,**0.999999713348428**,**0.999999981010438**,**0.999999999013412**,**0.99999999995984**]**

#Excel range of the Random Variable

Dose**=[**i**/**2 **for** i **in** **range(**23**)]**

#List of generated values by Created Object (obj)

Generated**=[**obj**.**GetProbability**(**i**)** **for** i **in** Dose**]**

plt**.**figure**()**

plt**.**plot**(**Dose**,**Excel**,**label**=**'Excel'**)**

plt**.**plot**(**Dose**,**Generated**,**label**=**'Generated'**)**

plt**.**legend**()**

plt**.**xlabel**(**'Dose'**)**

plt**.**ylabel**(**'Probability'**)**

plt**.**title**(**f'CDF Distribution for distribution={obj**.**DistType} mean={obj**.**mean} and std={obj**.**StdDev}'**,**color**=**'g'**)**

plt**.**show**()**

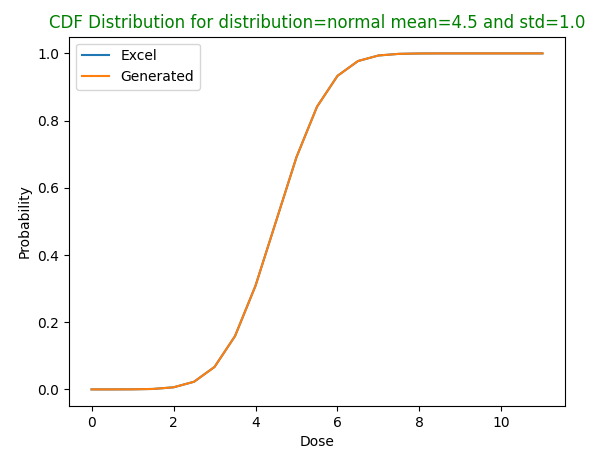


Figure - Probit Normal Distribution Verification

And same approach for lognormal distribution:

#Verification for log normal distribution

opr**.**wipe**()**

obj**=**opr**.**Fragilities**.**Probit**(**1**,**Distribution\_Type**=**'lognormal'**,** K1**=**1**,** K2**=**0.5**)**

#Excel results for mean=5 and std=2

Excel**=[**4.30656462230891E-20**,**0.0139954138003093**,**0.0662562227267046**,**0.135927084948079**,**0.208648223994374**,**0.278273867277455**,**0.342497894582516**,**0.400711942041258**,**0.453044380936582**,**0.499924069447986**,**0.541879616184157**,**0.579447543942368**,**0.613132088240471**,**0.643389800296413**,**0.670626009729306**,**0.695196771402629**,**0.717413125048509**,**0.737546096844058**,**0.755831684715014**,**0.772475484841812**,**0.787656828725862**,**0.801532406011979**,**0.8142393985932**,**0.8258981728861**,**0.836614583647308**,**0.846481941822576**,**0.855582694679972**,**0.863989860981988**,**0.871768258271563**,**0.878975554000964**,**0.885663167438306**,**0.891877045105856**,**0.897658328920541**,**0.903043933167712**,**0.908067043878904**,**0.912757552036115**,**0.917142430226971**,**0.921246060871349**,**0.925090522882637**,**0.928695842574893**,**0.932080213746174**,**0.935260191129611**,**0.938250860783288**,**0.941065990467936**,**0.943718162621335**,**0.946218892166573**,**0.948578731076635**,**0.950807361350948**,**0.952913677832627**,**0.954905862101998**,**0.956791448517015**]**

#Excel range of the Random Variable

Dose**=[**0.01**]+[**i**\***10 **for** i **in** **range(**1**,**51**)]**

#List of generated values by Created Object (obj)

Generated**=[**obj**.**GetProbability**(**i**)** **for** i **in** Dose**]**

plt**.**figure**()**

plt**.**plot**(**Dose**,**Excel**,**label**=**'Excel'**)**

plt**.**plot**(**Dose**,**Generated**,**label**=**'Generated'**)**

plt**.**legend**()**

plt**.**xlabel**(**'Dose'**)**

plt**.**ylabel**(**'Probability'**)**

plt**.**title**(**f'CDF Distribution for distribution={obj**.**DistType} mean={obj**.**mean} and std={obj**.**StdDev}'**,**color**=**'g'**)**

plt**.**show**()**

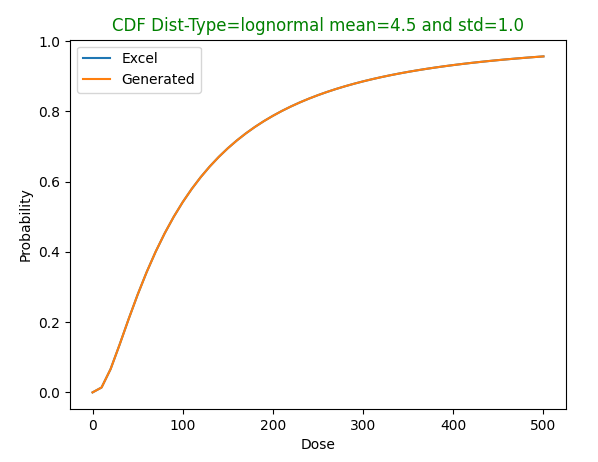


Figure - Probit Lognormal Distribution Verification

As it is seen on Figure 8 and Figure 9 the results are fully compatible with results calculated by MS Excel.

## OutFlowModels

### GasUnitHole Module

To model the outflow of gases from a hole, the algorithm shown in Flowchart 1 has been hired. is the internal pressure and is the outside pressure and the other parameters can be found from mentioned reference. The reference that is considered here is the Casal book and the following flowchart is exported from mentioned book section 2.3.1.2 to model the outflow rate of gases from a hole on the body of a plant unit. Using following code, the results will be verified:

# Verify example

opr**.**wipe**()**

#Define Site properties

opr**.**Sites**.**Site**(**tag**=**1**,** Temperature**=**20**+**273**,** Pressure**=**1.013**\***10**\*\***5**,** g**=**9.81**,** OngroundTemprature**=**20**,** Airdensity**=**1.21**)**

#define substance

propane**=**1

opr**.**Substance**.**Material**(**propane**,**Specific\_Heat\_Ratio**=**1.15**,**Molecular\_Weight**=**44.1**/**1000**,**GasDensity**=**1.808**)**

#define outflow model

OutFlowObj1**=**opr**.**OutFlowModel**.**GasUnitHole**(**1**,** Hole\_Diameter**=**0.02**,** Total\_t**=**20**,** Cd**=**0.62**,** Gas\_Constant**=**8.31446261815324**,)**

OutFlowObj2**=**opr**.**OutFlowModel**.**GasUnitHole**(**2**,** Hole\_Diameter**=**0.02**,** Total\_t**=**20**,** Cd**=**1**,** Gas\_Constant**=**8.31446261815324**,)**

#define plant unit object

UniObj**=**opr**.**PlantUnits**.**ONGStorage**(** tag**=**1**,** SiteTag**=**1**,** SubstanceTag**=**1**,** Pressure**=**10**\***10**\*\***5**,** Temperature**=**25**+**273**,** SubstanceVolumeRatio**=**1**,**

Diameter**=**10**,** Height**=**10**)**

#Assign outflow object to the plant unit handy and call the calculation method

UniObj**.**OutFlowModelObject**=**OutFlowObj1

UniObj**.**OutFlowModelObject**.**UnitObject**=**UniObj

UniObj**.**OutFlowModelObject**.**Calculate**()**

#get some of the calculation results

mdot**=**UniObj**.**OutFlowModelObject**.**MassGasReleaseRate

mTotal=UniObj.OutFlowModelObject.TotalMassGas\_Release

**print(**'For Cd=0.62 the mass outflow rate = '**,** mdot**[**1**] ,**' and the total released mass ='**,** mTotal[-1]**)**

#Check for second case

UniObj**.**OutFlowModelObject**=**OutFlowObj2

UniObj**.**OutFlowModelObject**.**UnitObject**=**UniObj

UniObj**.**OutFlowModelObject**.**Calculate**()**

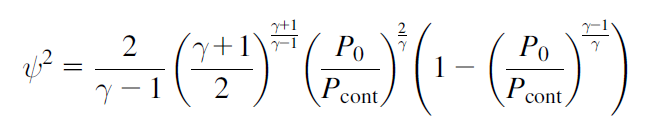
mdot**=**UniObj**.**OutFlowModelObject**.**MassGasReleaseRate

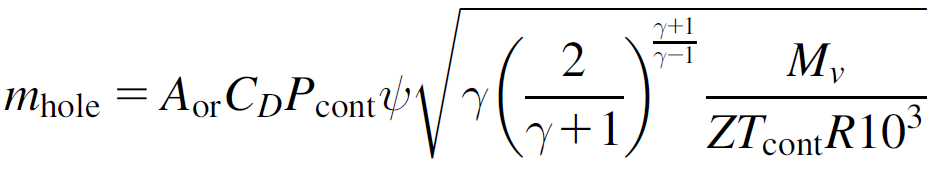
mTotal=UniObj.OutFlowModelObject.TotalMassGas\_Release

**print(**'For Cd=1 the mass outflow rate = '**,** mdot**[**1**]** **,**' and the total released mass ='**,** mTotal[-1]**)**

**print(**'Results are compatible with Casal results on example 2.3 that for cd=0.62 dm=0.525 and for cd=1 dm=0.847'**)**

Input Data





*No*

*Yes*

Flowchart - Gas OutFlow Algorithm

The results of the above code are compatible with Casal examples:

For Cd=0.62 the mass outflow rate = 0.5248 and the total released mass = 10.496

For Cd=1 the mass outflow rate = 0.8464 and the total released mass = 33.8568

Results are compatible with Casal results on example 2.3 that for cd=0.62 md=0.525 and for cd=1 md=0.847

### TankHole Family

TankHole family is a family of the cases that consider the outflow of the liquid from a hole in the body of the plant unit. But in any module of this family, the duration or approach of calculations varies but total algorithm is same. To find out more about the details of the calculations you can check Casal book chapter 2.

#### TankHole module

To verify the results of the implemented algorithm in this module, with the results of the module will be check by example no 2.1 of Casal book. In this module, the calculated released volume is equal to the total volume of the substance that is above the plant unit body hole. The following code model the condition explained in the mentioned book and draw the results and compare them with given results in the mentioned example:

#Verification with Casal example 2.1

opr**.**wipe**()**

SiteTAg**=**1

opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**20**+**273**,**Pressure**=**1**\***10197.16 **,**XSiteBoundary**=[**0**,**100**,**100**,**0**],** YSiteBoundary**=[**0**,**0**,**100**,**100**],** g**=**9.81**)**

#Define Material

opr**.**Substance**.**DataBank**.**CasCalEx2\_1**(**1**)**

#Define Plant Unit

Uni**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**Pressure**=**1**\***10197.16 **,**Temperature**=**2**,**Diameter**=**5**,**Height**=**10**,**

SubstanceTag**=**1**,**SubstanceVolumeRatio**=**0.85**)** #Define a StorageTank

# define outflow model object

OutFlowObj**=**opr**.**OutFlowModel**.**TankHole**(**1**,**Hole\_Diameter**=**0.05**,** Hole\_Height\_FromBot**=**1**,** delta\_t**=**180**,** Cd**=**0.62**,);**

OutFlowObj**.**UnitObject**=**Uni

OutFlowObj**.**Calculate**()**

MassRateG**=**OutFlowObj**.**MassLiquidReleaseRate

MassTotal**=**OutFlowObj**.**TotalMassLiquid\_Release

t**=**OutFlowObj**.**t\_release

#Casal Example 2.1 Results

TotalMass**=[**0**,**2304**,**4586**,**6850**,**9092**,**11313**,**13514**,**15694**,**17853**,**19991**,**22107**]**

MassRate**=[**0**,**12.8**,**12.68**,**12.57**,**12.45**,**12.34**,**12.23**,**12.11**,**11.99**,**11.88**,**11.76**]**

plt**.**figure**()**

plt**.**plot**(**t**[**0**:len(**TotalMass**)],**TotalMass**,** label**=**'Casal Results'**)**

plt**.**plot**(**t**[**0**:len(**TotalMass**)],**MassTotal**[**0**:len(**TotalMass**)],**label**=**'Generated'**)**

plt**.**title**(**'Total mass release'**)**

plt**.**xlabel**(**'time (s)'**)**

plt**.**ylabel**(**'Total mass released (kg)'**)**

plt**.**legend**()**

plt**.**figure**()**

plt**.**plot**(**t**[**0**:len(**MassRate**)],**MassRate**,** label**=**'Casal Results'**)**

plt**.**plot**(**t**[**0**:len(**MassRate**)],**MassRateG**[**0**:len(**MassRate**)],**label**=**'Generated'**)**

plt**.**legend**()**

plt**.**title**(**'Release Mass Rate '**)**

plt**.**xlabel**(**'time (s)'**)**

plt**.**ylabel**(**'Mass Release rate (kg/2)'**)**

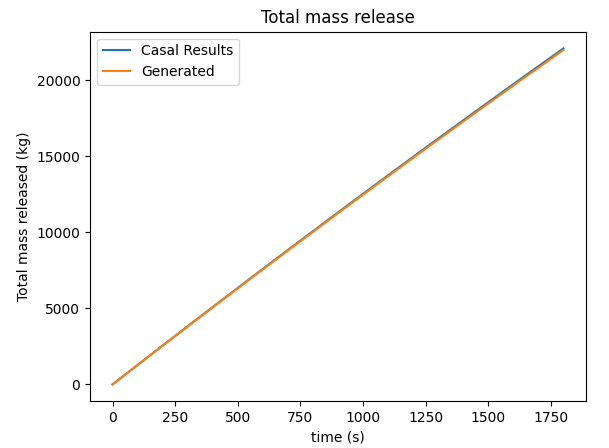


Figure - Verification on Total Released Mass for TankHole Module

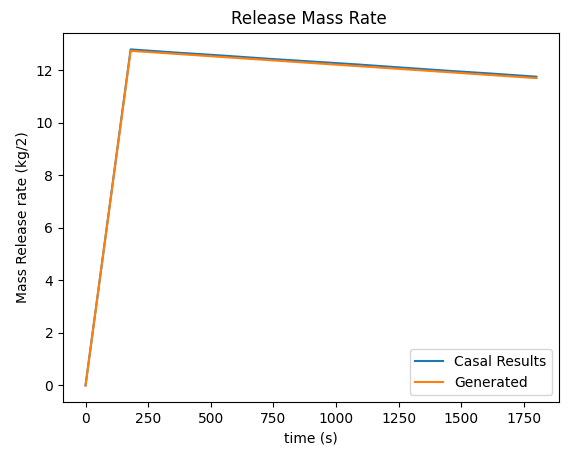


Figure - Verification on Released Mass Rate for TankHole module

As it is shown on the Figure 10 and Figure 11 the results are properly compatible on the handy calculated results.

#### TankHoleDuration module

This module is same TankHole module but after entered duration the calculation will stop and no released volume is corresponding to the volume that has been calculated until the entered duration. Results are compatible with previous results and it has not been plotted again.

#### TankHoleFixStep module

This module is again same TankHole module but after entered duration the calculation will stop and no released volume is corresponding to the volume that has been calculated until the entered duration. Results are compatible with previous results and it has not been plotted again.

#### TankHoleInitRate module

In this module the mass outflow rate considers constant and equal to the initial rate. Simply by a handy Calculation the outflow mass can be calculated and check by the code results. It is checked by the following code:

#Verification with Casal example 2.1 for TankHoleInitRate

opr**.**wipe**()**

SiteTAg**=**1

opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**20**+**273**,**Pressure**=**1**\***10197.16 **,**XSiteBoundary**=[**0**,**100**,**100**,**0**],** YSiteBoundary**=[**0**,**0**,**100**,**100**],** g**=**9.81**)**

#Define Material

opr**.**Substance**.**DataBank**.**CasCalEx2\_1**(**1**)**

#Define Plant Unit

Uni**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**Pressure**=**1**\***10197.16 **,**Temperature**=**2**,**Diameter**=**5**,**Height**=**10**,**

SubstanceTag**=**1**,**SubstanceVolumeRatio**=**0.85**)** #Define a StorageTank

# define outflow model object

OutFlowObj**=**opr**.**OutFlowModel**.**TankHoleInitRate**(**1**,**Hole\_Diameter**=**0.05**,** Hole\_Height\_FromBot**=**1**,** delta\_t**=**180**,** Cd**=**0.62**);**

OutFlowObj**.**UnitObject**=**Uni

OutFlowObj**.**Calculate**()**

MassRateG**=**OutFlowObj**.**MassLiquidReleaseRate

MassTotal**=**OutFlowObj**.**TotalMassLiquid\_Release

t**=**OutFlowObj**.**t\_release

Handymass**=[**MassRateG**[**0**]\***i **for** i **in** t**]**

# print(t,MassTotal)

#Casal Example 2.1 Results

TotalMass**=[**0**,**2304**,**4586**,**6850**,**9092**,**11313**,**13514**,**15694**,**17853**,**19991**,**22107**]**

Casalt**=[**i**\***180 **for** i **in** **range(len(**TotalMass**))]**

MassRate**=[**0**,**12.8**,**12.68**,**12.57**,**12.45**,**12.34**,**12.23**,**12.11**,**11.99**,**11.88**,**11.76**]**

plt**.**figure**()**

plt**.**plot**(**Casalt**,**TotalMass**,** label**=**'Casal Results'**)**

plt**.**plot**(**t**,**MassTotal**,**label**=**'Generated'**)**

plt**.**plot**(**t**,**Handymass**,**label**=**'Handy OutFlow MAss'**)**

plt**.**title**(**'Total mass release'**)**

plt**.**xlabel**(**'time (s)'**)**

plt**.**ylabel**(**'Total mass released (kg)'**)**

plt**.**legend**()**

plt**.**figure**()**

plt**.**plot**(**Casalt**,**MassRate**,** label**=**'Casal Results'**)**

plt**.**plot**(**t**,**MassRateG**,**label**=**'Generated'**)**

plt**.**legend**()**

plt**.**title**(**'Release Mass Rate '**)**

plt**.**xlabel**(**'time (s)'**)**

plt**.**ylabel**(**'Mass Release rate (kg/2)'**)**

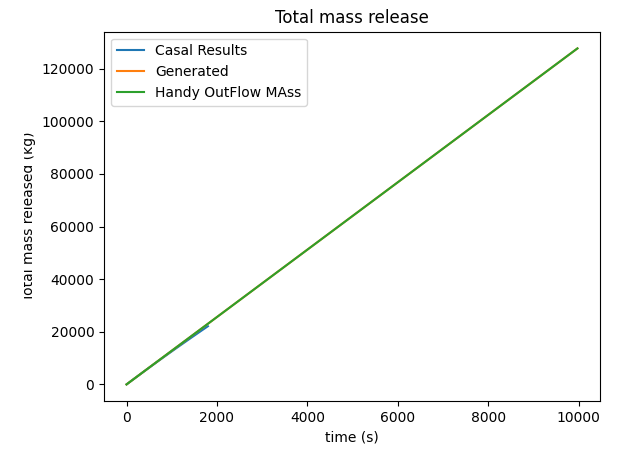


Figure - Verification on Total Released Mass for TankHoleInitRate Module

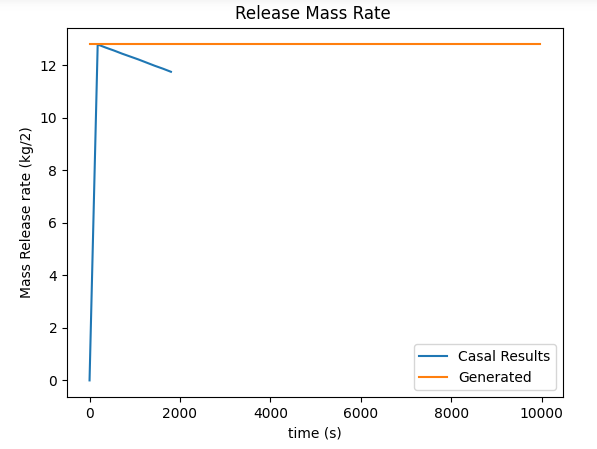


Figure - Verification on Released Mass Rate for TankHoleInitRate module

As it is seen in the Figure 13 the initial calculated rate is compatible with the Casal Book example and it is considered constant by the code, and the results of the outflow that has been shown on the Figure 12 is completely compatible on the handy calculations.

### Simultaneous Family

In this family for two cases of the content state (Liquid – Gas) and models content outflow in a very small duration (0.01 sec)

#### SimultaniousLiquid module

#### SimultaniousGas module

Both o the above modules because of the simplicity of the calculations, do not need any verification and it returns total mass of the content as the released mass and the duration of the release has been considered equal to 0.01 seconds.

### NoOutFlow Module

As it is obvious from the module name, this module considers no outflow and it returns zero for all outflow parameters and there is no need for any verification.

## Connectors

The main response of the connectors is to make decisions for the following consequences according defined probability distribution. So, what should be verified for them is the accuracy of making decisions or selecting the following consequences according defined probability distribution.

### DS\_LOC module

By the following code this accuracy has been checked:

#Verify Distribution

opr**.**wipe**()**

#Define Some Imaginary Outflow Models

tag**=**1

opr**.**OutFlowModel**.**TankHole**(**tag**,** Hole\_Diameter**=**0.05**,** Hole\_Height\_FromBot**=**0**,** delta\_t**=**100**,** Cd**=**1**)**

opr**.**OutFlowModel**.**TankHole**(**2**,** Hole\_Diameter**=**0.01**,** Hole\_Height\_FromBot**=**1**,** delta\_t**=**150**,** Cd**=**0.62**)**

opr**.**OutFlowModel**.**TankHole**(**3**,** Hole\_Diameter**=**0.5**,** Hole\_Height\_FromBot**=**0.5**,** delta\_t**=**100**,** Cd**=**0.62**)**

opr**.**OutFlowModel**.**Liquid10min**(**4**)**

opr**.**OutFlowModel**.**Liquid10min**(**5**)**

#Define a DS\_LOC

DSLOCOBJ**=**opr**.**DS\_LOC**(**tag**=**1**,**FragilityTag**=**1**,**OutFlowModelTagList**=[**1**,**4**,**3**],**LOCProbabilityList**=[**6**,**3**,**1**])**

Generated**=[]**

**for** i **in** **range(**100**):**

N**=**1000

rslt**=dict()**

rslt**={**i**:**0 **for** i **in** DSLOCOBJ**.**OutFlowModelTagList**}**

**for** i **in** **range(**N**):**

obj**=**DSLOCOBJ**.**Give1OutFlowModel**()**

rslt**[**obj**.**tag**]=**rslt**[**obj**.**tag**]+**1

**for** i**,**j **in** rslt**.**items**():**

rslt**[**i**]=**j**/**N

Generated**.**append**(**rslt**)**

plt**.**figure**()**

plt**.**plot**([**0.6 **for** i **in** Generated**],** color**=**'green'**,**linestyle**=**'dashed'**)**

plt**.**plot**([**0.3 **for** i **in** Generated**],** color**=**'green'**,**linestyle**=**'dashed'**)**

plt**.**plot**([**0.1 **for** i **in** Generated**],** color**=**'green'**,**linestyle**=**'dashed'**)**

plt**.**plot**([**i**[**1**]** **for** i **in** Generated**],** label**=**'Reults for Fragility 1 with Probability 0.6'**)**

plt**.**plot**([**i**[**4**]** **for** i **in** Generated**],** label**=**'Reults for Fragility 4 with Probability 0.3'**)**

plt**.**plot**([**i**[**3**]** **for** i **in** Generated**],** label**=**'Reults for Fragility 3 with Probability 0.1'**)**

plt**.**title**(**'Resutls of 1000 sampling in 100 repeating test'**)**

plt**.**xlabel**(**'Test Number'**)**

plt**.**ylabel**(**'Frequency of Occurance'**)**

The results for 1000 times sampling show proper compatibility on the defined probability values as shown in Figure 14.



Figure - Results of DS-LOC Connector Sampling

### Pb\_LOC module

Same approach for DS\_LOC implemented for this module and same results reached and because of the similarity of the modules the corresponding code is not written.

### Out\_Physic module

Same approach for DS\_LOC implemented for this module and same results reached and because of the similarity of the modules the corresponding code is not written.

## DispersionSpreadModels

As it is obvious, this subpackage is responsible for calculations of mass dispersion in the environment. For liquids and gases there are different approaches and formulations and different modules are available for this purpose. In the following current modules are investigated and verified.

### LiquidSpread Module

In this module the outflow mass will be dispersed in a circular shape with the center equal to the unit and radius equal to dike area if defined or radius equal to volume of the outflow mass and defined thickness.

This module is responsible for check the availability of dike and limiting the spread of liquid to the dike area and if there is no dike, determine the minimum thickness of the liquid depend on the defined surface toughness and return the area (radius) of the spread liquid.

To check the functionality of this module to handle the above responsibilities, using the following code, results has been checked:

#Verification for Pool Liquid Spread

opr**.**wipe**()**

SiteTAg**=**1

opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**40**,**Pressure**=**1**\***10197.16**)**

#Define Material

Subsobj**=**opr**.**Substance**.**DataBank**.**CasCalEx2\_1**(**tag**=**1**)**

#Define Outflow Models

OutFlowObj**=**opr**.**OutFlowModel**.**TankHole**(**tag**=**1**,** Hole\_Diameter**=**0.1**,** Hole\_Height\_FromBot**=**0**,** delta\_t**=**100**,** Cd**=**1**)**

#Define Dispesion Spread Models and their connections to the materials and outflows

DispObject**=**opr**.**DispersionSpreadModels**.**LiquidSpread**(**tag**=**1**,** MatTags**=[**1**],** OutFlowModelTags**=[**1**],**

MinDisThickness**=**0.005**,**Surface\_Roughnesslist**=[**0.1**,**0.2**,**0.3**],**

Surface\_RoughnessThickness**=[**0.01**,**0.015**,**0.02**])**

#Define Plant Unit

UnitObj**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**SiteTag**=**SiteTAg**,**

Horizontal\_localPosition**=**10**,**Vertical\_localPosition**=**15**,**

Diameter**=**5**,**Height**=**10**,**

SubstanceTag**=**1**,**SubstanceVolumeRatio**=**0.85**,**Surface\_Roughness**=**0.2**)**

#Handy Calculations

VSubs**=**3.1415**\***5**\*\***2**/**4**\***10**\***0.85

MassSub**=**Subsobj**.**Density**\***VSubs

PoolArea**=**VSubs**/**0.015

PoolRadius**=(**PoolArea**\***4**/**3.1415**)\*\***0.5**/**2

#These steps will be done Automatically By the Program inside the Analysis Part

UnitObj**.**OutFlowModelObject**=**deepcopy**(**OutFlowObj**)** #Assign OutFlow Model to Unit Object

UnitObj**.**OutFlowModelObject**.**UnitObject**=**UnitObj

UnitObj**.**OutFlowModelObject**.**Calculate**();** #Calculate OutFlow Calculations to get

UnitObj**.**DispersionModelObject**=**DispObject #Assign Dispersion Object to the Unit Object

UnitObj**.**DispersionModelObject**.**UnitObject**=**UnitObj #Assign the UnitObject to the Dispersion Model

UnitObj**.**DispersionModelObject**.**Calculate**()** #Do Calculations

**print(**'\n Dispersion Time='**,**UnitObj**.**DispersionModelObject**.**t\_disp**)**

**print(**'\n Liquid Dispersion Center at each Time='**,**UnitObj**.**DispersionModelObject**.**LiquidCenter**)**

**print(**'\n Liquid Dispersion Radious ='**,max(**UnitObj**.**DispersionModelObject**.**LiquidRadious**),** 'And handy Calculated Radius is='**,**PoolRadius**)**

**print(**'\n Liquid Dispersion Thickness ='**,min(**UnitObj**.**DispersionModelObject**.**LiquidThickness**))**

The Following results of the above code shows that the calculations for a liquid pool spread is compatible with handy calculations and considered hypothesizes.

Dispersion Time**=** **[**0**,** 100**,** 200**,** 300**,** 400**,** 500**,** 600**,** 700**,** 800**,** 900**,** 1000**,** 1100**,** 1200**,** 1300**,** 1400**,** 1500**,** 1600**,** 1700**,** 1800**,** 1900**,** 2000**,** 2100**,** 2200**,** 2300**,** 2400**,** 2500**,** 2600**,** 2700**,** 2800**,** 2900**,** 3000**,** 3100**,** 3200.5558851357086**]**

Liquid Dispersion Center at each Time**=** **[(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**),** **(**10**,** 15**)]**

Liquid Dispersion Radious **=** 59.51278116960603 And handy Calculated Radius **is=** 59.511903571190416

Liquid Dispersion Thickness **=** 0.015

To check the effect of the dike, above example has been repeated by considering a dike with area and radius less than above calculated area:

#Verification for Pool Liquid Spread With considering a dike

opr**.**wipe**()**

SiteTAg**=**1

opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**40**,**Pressure**=**1**\***10197.16**)**

#Define Material

Subsobj**=**opr**.**Substance**.**DataBank**.**CasCalEx2\_1**(**tag**=**1**)**

#Define Outflow Models

OutFlowObj**=**opr**.**OutFlowModel**.**TankHole**(**tag**=**1**,** Hole\_Diameter**=**0.1**,** Hole\_Height\_FromBot**=**0**,** delta\_t**=**100**,** Cd**=**1**)**

#Define Dispesion Spread Models and their connections to the materials and outflows

DispObject**=**opr**.**DispersionSpreadModels**.**LiquidSpread**(**tag**=**1**,** MatTags**=[**1**],** OutFlowModelTags**=[**1**],**

MinDisThickness**=**0.005**,**Surface\_Roughnesslist**=[**0.1**,**0.2**,**0.3**],**

Surface\_RoughnessThickness**=[**0.01**,**0.015**,**0.02**])**

#Define Dike Object

Adike**=**5000

opr**.**Safety**.**Dike**(**tag**=**1**,**Height**=**2**,** Area**=**Adike**)**

#Define Plant Unit

UnitObj**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**SiteTag**=**SiteTAg**,**DikeTag**=**1**,**

Horizontal\_localPosition**=**10**,**Vertical\_localPosition**=**15**,**

Diameter**=**5**,**Height**=**10**,**

SubstanceTag**=**1**,**SubstanceVolumeRatio**=**0.85**,**Surface\_Roughness**=**0.2**)**

#Handy Calculations

VSubs**=**3.1415**\***5**\*\***2**/**4**\***10**\***0.85

MassSub**=**Subsobj**.**Density**\***VSubs

PoolRadius**=(**Adike**\***4**/**3.1415**)\*\***0.5**/**2

#These steps will be done Automatically By the Program inside the Analysis Part

UnitObj**.**OutFlowModelObject**=**deepcopy**(**OutFlowObj**)** #Assign OutFlow Model to Unit Object

UnitObj**.**OutFlowModelObject**.**UnitObject**=**UnitObj

UnitObj**.**OutFlowModelObject**.**Calculate**();** #Calculate OutFlow Calculations to get

UnitObj**.**DispersionModelObject**=**DispObject #Assign Dispersion Object to the Unit Object

UnitObj**.**DispersionModelObject**.**UnitObject**=**UnitObj #Assign the UnitObject to the Dispersion Model

UnitObj**.**DispersionModelObject**.**Calculate**()** #Do Calculations

**print(**'\n Dispersion Time='**,**UnitObj**.**DispersionModelObject**.**t\_disp**)**

**print(**'\n Liquid Dispersion Center at each Time='**,**UnitObj**.**DispersionModelObject**.**LiquidCenter**)**

**print(**'\n Liquid Dispersion Radius ='**,max(**UnitObj**.**DispersionModelObject**.**LiquidRadious**),** 'And handy Calculated Radius is='**,**PoolRadius**)**

The results are compatible with handy calculations:

Dispersion Time= [0, 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 1900, 2000, 2100, 2200, 2300, 2400, 2500, 2600, 2700, 2800, 2900, 3000, 3100, 3200.5558851357086]

Liquid Dispersion Center at each Time= [(10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15), (10, 15)]

Liquid Dispersion Radius = 39.89481634448608 And handy Calculated Radius is= 39.89481634448608

Above time steps are the outflow time steps.

### BritterMcQuaid Module

This module is used for modeling dispersion of the gas. Because of its analytical formulation and fast calculation is a proper model for using in this platform. To check its validity, the results of this model is checked by an example that is provided in Casal book. (Example 7.5)

opr**.**wipe**()**

opr**.**Sites**.**Site**(**tag**=**1**,** Temperature**=**20**+**273**,** Pressure**=**1.013**\***10**\*\***5**,** g**=**9.81**,** OngroundTemprature**=**20**,** Airdensity**=**1.21**)**

windobj**=**opr**.**WindData**.**WindRose**(**1**)**

windobj**.**WindDirection**=**90

windobj**.**WindSpeed**=**5.2

propane**=**1

opr**.**Substance**.**Material**(**propane**,**Density**=**553**,**GasDensity**=**1**,** BoilingPointGasDensity**=**2.32**,** Specific\_Heat\_Ratio**=**1.15**,**Molecular\_Weight**=**44.1**/**1000**,**

Lower\_Flammability\_Limit**=**0.3**,**Upper\_Flammability\_Limit**=**0.7**)**

OutFlowObj**=**opr**.**OutFlowModel**.**GasUnitHole**(**1**,** Hole\_Diameter**=**0.02**,** Total\_t**=**20**,** Cd**=**1**,** Gas\_Constant**=**8.31446261815324**)**

DispObj**=**opr**.**DispersionSpreadModels**.**BritterMcQuaid**(**1**,** MatTags**=[**1**],** OutFlowModelTags**=[**1**],)**

UniObj**=**opr**.**PlantUnits**.**ONGStorage**(**tag**=**1**,** SiteTag**=**1**,** SubstanceTag**=**1**,** FragilityTagNumbers**=[**1**],** Horizontal\_localPosition**=**0**,** Vertical\_localPosition**=**0**,** Pressure**=**375**\***10**\*\***5**,** Temperature**=-**42**+**273**,**

SubstanceVolumeRatio**=**1**,** Diameter**=**10**,** Height**=**10**)**

#Do the Analysis Part Handy

UniObj**.**OutFlowModelObject**=**OutFlowObj

UniObj**.**OutFlowModelObject**.**UnitObject**=**UniObj

UniObj**.**OutFlowModelObject**.**Calculate**()**

# print(mdot[0])

UniObj**.**DispersionSpreadModelObject**=**DispObj

UniObj**.**DispersionSpreadModelObject**.**UnitObject**=**UniObj

UniObj**.**DispersionSpreadModelObject**.**Calculate**()**

#To get Concentration at a specific distance

**print(**'Concentration at distance 340 m is='**,**UniObj**.**DispersionSpreadModelObject**.**GasConcentration**(**340**,**0**,**0**))**

**print(**'The result is compatible with what is solved in cascal book example 7.5 for LFL=2.1%'**)**

#Get Distance Corresponding to a Concentration

**print(**'Distance Corresponding to the Concentration 0.021 is='**,**UniObj**.**DispersionSpreadModelObject**.**Concdist**(**0.021**)[**0**])**

**print(**'The result is compatible with what is solved in cascal book example 7.5 for LFL=2.1% that is 340 m'**)**

The results are compatible with handy calculations:

Concentration at distance 340 m is= 0.019549915257768426

The result is compatible with what is solved in Casal book example 7.5 for LFL=2.1%

Distance Corresponding to the Concentration 0.021 is= 321.1696981693521

The result is compatible with what is solved in Casal book example 7.5 for LFL=2.1% that is 340 m

Some deviations are because of the numerical rounding that are acceptable.

### GasGuassian Module

This module is used for gas dispersion in the space. It uses the Gaussian formula and to check its validity again its results are compared with Casal bool example.

### LqdSprdGuassianGasDisp Module

This module is used for liquid substances that after outflow their evaporation should be calculated and the dispersion of the evaporated gas should be considered (explosive gas). For this purpose, liquid spread considered as same pool liquid with same parameters, and for gas evaporation rate and dispersion, the gaussian formula has been used. To validate the module calculations, by the below code the calculated results compared with Casal book example 2.9:

#Verification by casal example 2.9

opr**.**wipe**()**

SiteTAg**=**1

opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**20**+**273**,**Pressure**=**1.0132**\***10**\*\***5 **,**XSiteBoundary**=[**0**,**100**,**100**,**0**],** YSiteBoundary**=[**0**,**0**,**100**,**100**],** g**=**9.81**)**

#Create Wind Object

windobj**=**opr**.**WindData**.**WindRose**(**tag**=**1**)**

windobj**.**WindClass**=**"A"

windobj**.**WindDirection**=**90

windobj**.**WindSpeed**=**3

windobj**.**AlphaCOEF**=**1

windobj**.**isCalmn**=False**

#Define Material

subsObj**=**opr**.**Substance**.**DataBank**.**CasCalEx2\_1**(**1**)**

subsObj**.**Specific\_Heat\_of\_Vaporization**=**392.23**\***1000

subsObj**.**Vapour\_Pressure**=**16130 #Accoring Casal Example 2.9

subsObj**.**Molecular\_Weight**=**86**/**1000 #Accoring Casal Example 2.9

subsObj**.**Boiling\_Point**=**273**+**68.7 #According Casal Example 2.9

subsObj**.**Liquid\_Partial\_Pressure\_in\_Atmosphere**=**0 #According Casal Example 2.9

subsObj**.**Lower\_Flammability\_Limit**=**0.03

subsObj**.**Upper\_Flammability\_Limit**=**3

#Define Outflow Models

tag**=**1

opr**.**OutFlowModel**.**TankHole**(**tag**,** Hole\_Diameter**=**0.05**,** Hole\_Height\_FromBot**=**0**,** delta\_t**=**100**,** Cd**=**1**)**

opr**.**Safety**.**Dike**(**tag**=**1**,** Height**=**10**,** Area**=**3.1415**\***11**\*\***2**)**

#Define Plant Units

xc**=**0

yc**=**0

D**=**5

#OnGroundTag=1

Uni**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**SiteTag**=**SiteTAg**,**DikeTag**=**1**,**

Horizontal\_localPosition**=**xc**,**Vertical\_localPosition**=**yc**,**

Pressure**=**1.5**\***10**\*\***5 **,**Temperature**=**2**,**

Diameter**=**5**,**Height**=**10**,**

SubstanceTag**=**1**,**SubstanceVolumeRatio**=**0.85**)** #Define a StorageTank

Uni**.**isdamaged**=True**

#These steps will be done Automatically By the Program inside the Analysis Part

OutFlowObj**=**opr**.**OutFlowModel**.**ObjManager**[**1**]** # Get OutFlow Object

UnitObj**=**opr**.**PlantUnits**.**ObjManager**[**1**]** # Get Unit Object

Subs**=**opr**.**Substance**.**ObjManager**[**1**]** # Get Substance Object

SiteObj**=**opr**.**Sites**.**ObjManager**[**1**]** # Get Site Object

UnitObj**.**OutFlowModelObject**=**deepcopy**(**OutFlowObj**)** #Assign OutFlow Model to Unit Object

UnitObj**.**OutFlowModelObject**.**UnitObject**=**UnitObj

UnitObj**.**OutFlowModelObject**.**Calculate**();** #Calculate OutFlow Calculations to get

#Define Dispesion Spread Models and their connections to the materials and outflows

DispObject**=**opr**.**DispersionSpreadModels**.**LqdSprdGuassianGasDisp**(**tag**=**1**,**MatTags**=[**1**],** OutFlowModelTags**=[**1**,**2**,**3**,**4**])** #Creat the Dispersion Object

UnitObj**.**DispersionSpreadModelObject**=**DispObject #Assign Dispersion Object to the Unit Object

UnitObj**.**DispersionSpreadModelObject**.**UnitObject**=**UnitObj #Assign the UnitObject to the Dispersion Model

UnitObj**.**DispersionSpreadModelname**=**DispObject**.**name

#Do Calculations

DispObject**.**Calculate**();**

poolArea**=max(**DispObject**.**LiquidRadious**)\*\***2**\***3.1415

V**=**Uni**.**V\_subs

md**=max(**DispObject**.**LiquidVaporizationMassRate**)**

E**=**md**/**poolArea

h**=**V**/**poolArea

**print(**'The substance Height in Dike='**,**h**)**

**print(**'Evaporation of mass per area (E)='**,**E**)**

**print(**'Evaporation of mass md='**,**md**)**

**print(**'Results are compatible with Casal Example 2.9 that md=0.851'**)**

The results are compatible with handy calculations:

The substance Height in Dike= 0.43906253585224136

Evaporation of mass per area (E)= 0.002244769333550875

Evaporation of mass md= 0.853285086223359

Results are compatible with Casal Example 2.9 that md=0.851

The gaussian formula for gas dispersion is:

In the following to check the validation of code to be compatible with the gaussian formula, handy calculations using Excel will be done and will be compare with the code results. The weather and wind conditions are same as said in above example.

Code at first step calculates the center line of gas dispersion using \_CenterLine function. This line shows the maximum concentration of gas in each distance from the source. To check the results this center line values calculated separately using excel sheets and the results are as the following. As shown in Figure 15 the calculations are compatible on results of excel calculations.

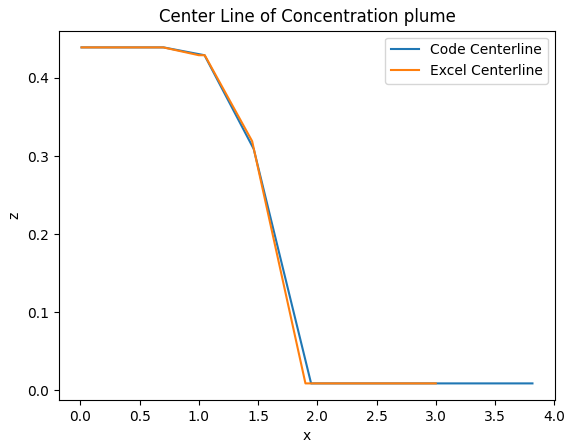


Figure - Validation on Centerline calculations

Gas Concentration at various dimensions also should be checked. Cross gas concentration at constant distance of source and constant height is also compared with excel results as shown in Figure 16.

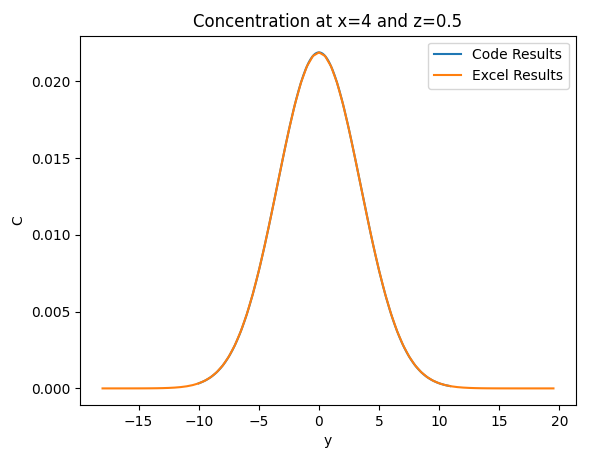


Figure - Validation on gas dispersion in cross dimension

Same validation done for longitudinal direction in a constant height and on ground. The results has been shown in Figure 17.

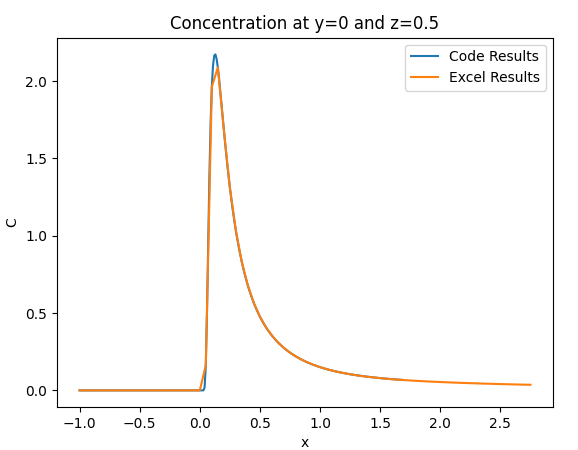


Figure - Validation on longitudinal direction

Also, code empowered to an algorithm to calculate the mass and its center of the dispersed gas. This algorithm uses the centerline values and for each center line, depend on its deviation in each cross direction, it selects number of points that covers the spaces that have concentration greater than LFL. By this way in a very short time the dispersed gas mass and its center will be calculated. To check the validation on the generated space and calculated mass and center, the following controls has been done using Excel:

* By considering the number of the segments equal to 1000 (a very huge number and 10 is enough and has enough accuracy) the maximum distance from center of the release becomes equal to:

The example LFL is equal to , so we expect that for any point greater than 3.5m we have no point greater than LFL. To check this condition for and on we check the all values on the z axis. The results are shown in Figure 18.

Figure - Variation of the Concentration along the height at different distances from source

As it can be seen all values on x=3.5 are less than LFL limitation and the maximum point on the ground is equal to LFL that shows we have no volume to be consider in mass calculations.

* For 1000 segments the calculated mass and center of the mass is equal to:

And for 10 segments it is equal to:

* To get ensure that Center of mass and mass are calculated correctly, for calculated points, the concentration are calculated and then center of the mass and mass also are calculated separately using excel and the results checks with code for 10 segments. As it is shown in Table 1 results are fully compatible as expected.

Table - Validation on Code Mass Calculations

|  |  |  |
| --- | --- | --- |
|  | Code | Excel |
| Mass () | 0.656 | 0.656 |
|  | - | 0.8737 |
|  | - | 0.251 |
|  | 1.332 | 1.332 |
|  | 0.382 | 0.382 |

* Calculated point for 10, 100, 500 and 1000 segments, also are shown in Figure 19, Figure 20, Figure 21 and Figure 22.

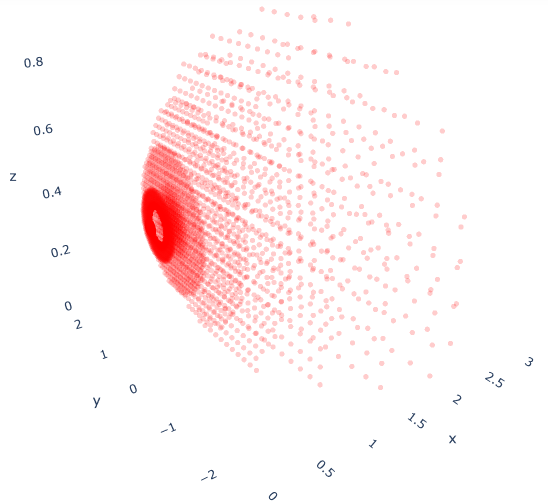


Figure - Gas Dispersion for 10 segments

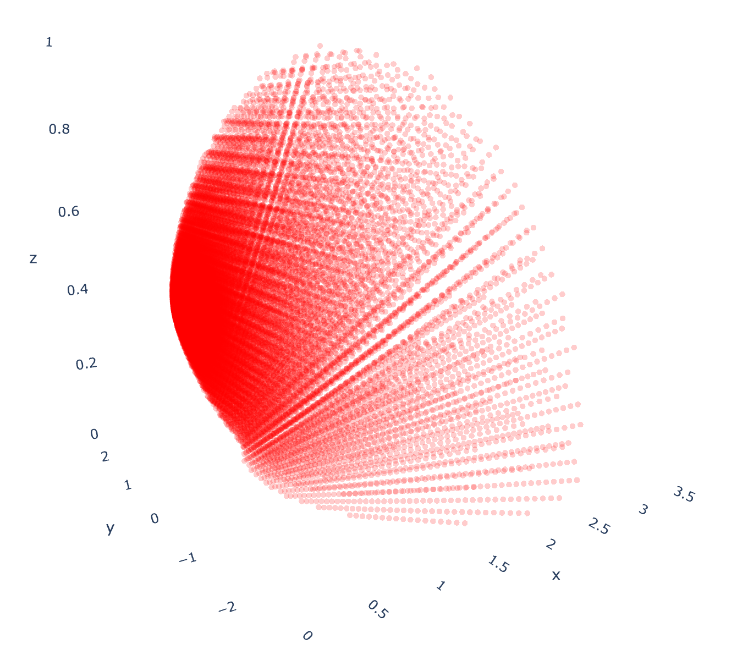


Figure - Gas Dispersion for 100 segments

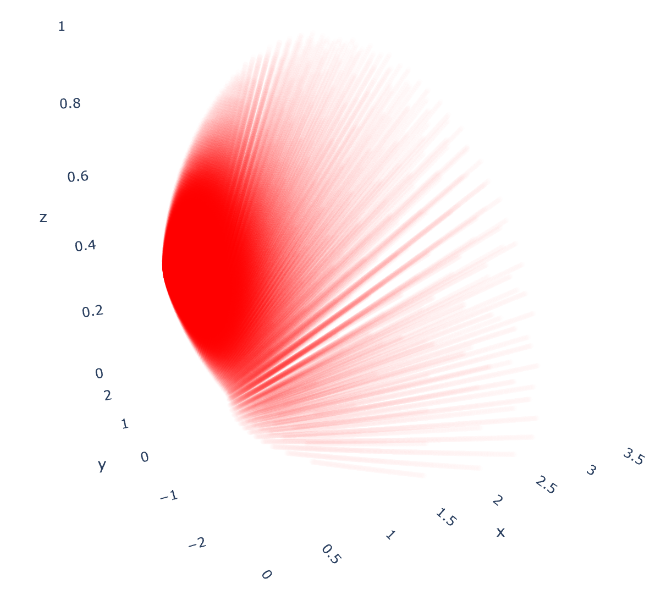


Figure - Gas Dispersion for 500 segments

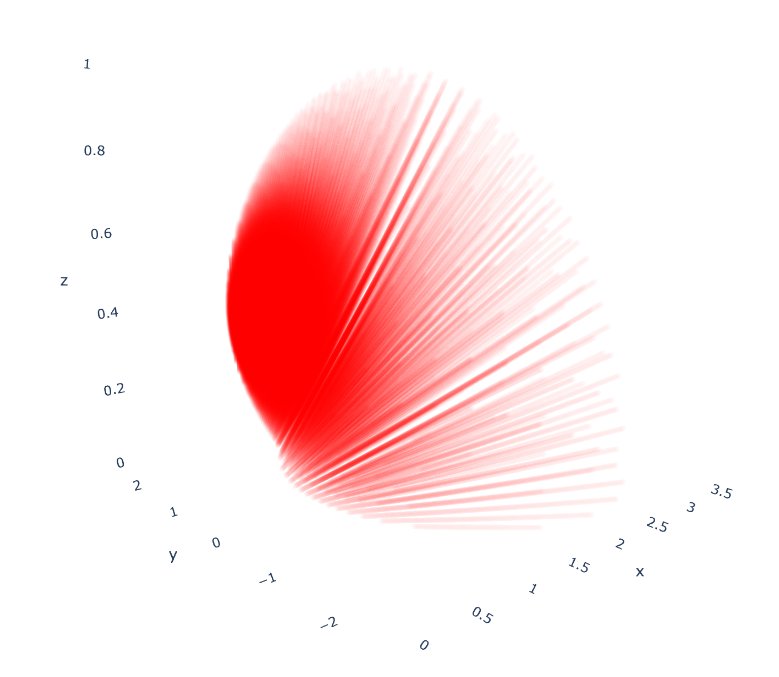


Figure - Gas Dispersion for 1000 segments

## PhysicalEffect SubPackage

### fire\_point\_source module

The fire point source model is formulated according point source model explained in Casal book. To validate the calculations of this module, example 3.4 of Casal book chose. The following code model the example condition and the final result is checked by example results:

opr**.**wipe**()**

#Define Wind Rose

windobj**=**opr**.**WindData**.**WindRose**(**1**)**

windobj**.**WindSpeed**=**0

windobj**.**WindDirection**=**90

#Define Site Condition and Geometry

SiteTAg**=**1

obj**=**opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**16**+**273**,** Pressure**=**1**\***10**\*\***5**,** XSiteBoundary**=[**0**,**100**,**100**,**0**],** YSiteBoundary**=[**0**,**0**,**100**,**100**],**

g**=**9.81**,**Humidity**=**0.79**,**Airdensity**=**0.270**)**

#Define Substance

subsobj**=**opr**.**Substance**.**DataBank**.**Butene**(**1**)** #Use DataBank to Load Material

subsobj**.**Specific\_Heat\_of\_Combustion**=**41900**\***1000

#Define OutFlow Model

OutflowObj**=**opr**.**OutFlowModel**.**TankHole**(**1**,** Hole\_Diameter**=**0.05**,** Hole\_Height\_FromBot**=**0**,** delta\_t**=**500**,** Cd**=**1**)**

#Define Dispersion Model

DispObj**=**opr**.**DispersionSpreadModels**.**LiquidSpread**(**tag**=**1**,** MatTags**=[**1**],** OutFlowModelTags**=[**1**],**MinDisThickness**=**0.005**,)**

#Define Dike for Tanks

opr**.**Safety**.**Dike**(**1**,**0**,**3.1415**\***6**\*\***2**/**4**)**

PlantObj**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**SiteTag**=**1**,**DikeTag**=**1**,** SubstanceTag**=**1**,** Diameter**=**10**,**

Horizontal\_localPosition**=**0**,** Vertical\_localPosition**=**0**,**

Height**=**20**,**SubstanceVolumeRatio**=**0.9**)**

#Manually define OutFlow Model for above Unit Object

PlantObj**.**OutFlowModelObject**=**OutflowObj

OutflowObj**.**UnitObject**=**PlantObj

PlantObj**.**OutFlowModelObject**.**Calculate**()**

#Manually define DispersionSpread Model for above Unit Object

PlantObj**.**DispersionSpreadModelObject**=**DispObj

PlantObj**.**DispersionSpreadModelObject**.**UnitObject**=**PlantObj

PlantObj**.**DispersionSpreadModelObject**.**Calculate**()**

obj**=**opr**.**PhysicalEffect**.**fire\_point\_source**(**1**,**minf**=**0.0501**,**k**=**1.5**)**

obj**.**UnitObject**=**PlantObj

**[**H**,**Hmax**,**D**,**Dprin**,**alpha**,**m**]=**obj**.**\_fireGeometry**()**

**[**Lp**,**d**,**phi**,**xf**,**yf**]=**obj**.**\_DistanceToFireCenterGeometry**(**18**,**0**,**1.6**)**

**print(**'H,D,m='**,round(**H**,**2**),round(**D**,**2**),round(**m**,**2**))**

**print(**'Lp,d,phi='**,round(**Lp**,**2**),round(**d**,**2**),round(**phi**,**2**))**

**print(**'Thermal Radiation intensity='**,**obj**.**Thermal\_Radiation\_at\_Point**(**18**,**0**,**1.6**)** **)**

Above code result is:

H,D,m= 11.5 6.0 0.05

Lp,d,phi= 18.47 15.39 0.23

Thermal Radiation intensity= 2878.505632375152

in cascal Example 3.4,

D=6 m

H=11.5

Point coordinate=[18,0,1.6]

Heat Of Combustion=41900\*10^3

Humidity=0.79

m=0.05

And the result Thermal Radiation intensity= 2800 W/m^2

As it is seen, the result is compatible with example calculations.

### VCE\_TNT module

To validate this module, the calculations are compared with Casal example 4.1. for this purpose, the following code has been provided:

opr**.**wipe**()**

#Define Wind Rose

windobj**=**opr**.**WindData**.**WindRose**(**1**)**

windobj**.**WindSpeed**=**0

windobj**.**WindDirection**=**90

#Define Site Condition and Geometry

SiteTAg**=**1

obj**=**opr**.**Sites**.**Site**(**SiteTAg**,** Temperature**=**16**+**273**,** Pressure**=**1.013**\***10**\*\***5**,** XSiteBoundary**=[**0**,**100**,**100**,**0**],** YSiteBoundary**=[**0**,**0**,**100**,**100**],**

g**=**9.81**,**Humidity**=**0.79**,**Airdensity**=**0.270**)**

#Define Substance

propane**=**1

subsobj**=**opr**.**Substance**.**Material**(**propane**,**name**=**'Propane'**,**Density**=**553**,**GasDensity**=**1**,**BoilingPointGasDensity**=**2.32**,** Specific\_Heat\_Ratio**=**1.15**,**

Molecular\_Weight**=**44.1**/**1000**,** Specific\_Heat\_of\_Combustion**=**43.930**\***10**\*\***6**,**

Lower\_Flammability\_Limit**=**0.2**,** Upper\_Flammability\_Limit**=**0.7**,)**

#Define OutFlow Model

OutflowObj**=**opr**.**OutFlowModel**.**GasUnitHole**(**tag**=**1**,** Hole\_Diameter**=**0.1**,** Total\_t**=**20**,** Cd**=**1**,** Gas\_Constant**=**8.31446261815324**)**

#Define Dispersion Model

DispObj**=**opr**.**DispersionSpreadModels**.**BritterMcQuaid**(**tag**=**1**,** MatTags**=**1**,** OutFlowModelTags**=[**1**],** MassParts**=**20**,)**

#Define Dike for Tanks

opr**.**Safety**.**Dike**(**1**,**0**,**3.1415**\***6**\*\***2**/**4**)**

#Define Plant Unit

PlantObj**=**opr**.**PlantUnits**.**ONGStorage**(**1**,**SiteTag**=**1**,**DikeTag**=**1**,** SubstanceTag**=**1**,** Diameter**=**10**,** Pressure**=** 1.769025**\***10**\*\***5**,** Temperature**=**16**+**273**,**

Horizontal\_localPosition**=**0**,** Vertical\_localPosition**=**0**,**

Height**=**20**,**SubstanceVolumeRatio**=**0.9**)**

#Manually define OutFlow Model for above Unit Object

PlantObj**.**OutFlowModelObject**=**OutflowObj

OutflowObj**.**UnitObject**=**PlantObj

PlantObj**.**OutFlowModelObject**.**Calculate**()**

#Manually define DispersionSpread Model for above Unit Object

PlantObj**.**DispersionSpreadModelObject**=**DispObj

PlantObj**.**DispersionSpreadModelObject**.**UnitObject**=**PlantObj

PlantObj**.**DispersionSpreadModelObject**.**Calculate**()**

**print(**"Exposive mass ="**,**DispObj**.**GasExplosiveMass**)**

**print(**"Explosive Mass center = "**,** DispObj**.**GasExplosiveCenterX**,**DispObj**.**GasExplosiveCenterY**,**DispObj**.**GasExplosiveCenterZ**)**

obj**=**opr**.**PhysicalEffect**.**VCE\_TNT**(**1**)**

obj**.**UnitObject**=**PlantObj

obj**.**Calculate**()**

dP**=**obj**.**OverPressure\_at\_Point**(**DispObj**.**GasExplosiveCenterX**[**0**]+**500**,**0**,**0**)**

**print(**"The Over Pressure Value in 500 m Over Explosive mass Center ="**,**dP**/**10**\*\***5**,** " Bar"**)**

The result of above code is as following:

Exposive mass = [30000.0545780576]

Explosive Mass center = [205.0062463385579] [0.0] [1.6781175585645267e-07]

The Over Pressure Value in 500 m Over Explosive mass Center = 0.048807472058782414 Bar

in casal Example 4.1,

d=500 m

M (Explosive Mass)= 30,000

DeltaHc=43930\*1000

And the result of over pressure is = 0.049 Bar

That is compatible with the above results!

### Safe module

There is no calculation to valid it.