Dokumen Pengembangan TRIAC
(TRIso *Analysis Code*)

LABORATORIUM KOMPUTASI

PUSAT TEKNOLOGI DAN KESELAMATAN REAKTOR NUKLIR

Disusun oleh:
Arya Adhyaksa Waskita

Supervisor:
Dr. Eng. Topan Setiadipura

06–09–2018

Daftar Isi

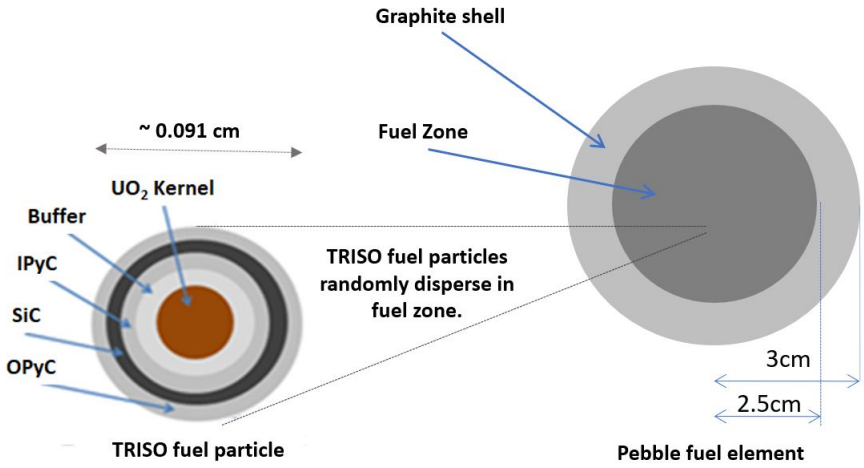
Daftar Gambar

Daftar Program

BAB 1

Pendahuluan

BATAN saat ini tengah berencana membangun reaktor riset baru berbasis HTGR (*High Temperature Gas-cooled Reactor*) [1] sebagai persiapan PLTN, yang akan dibangun di Indonesia di masa depan [2]. Salah satu yang perlu diperhatikan dalam pengembangan reaktor jenis ini adalah bahan bakarnya yang berjenis *pebble* yang bentuknya dapat diilustrasikan seperti pada Gambar 1.1. Bahan bakar harus dirancang sedemikian rupa sehingga rasio gagalnya bahan bakar selama operasi minimal.



Gambar 1.1: Ilustrasi bentuk bahan bakar *pebble* [3]

Bahan bakar berjenis *pebble* ini memiliki komponen utama yang dalam Gambar 1.1 disebut sebagai *triso fuel particle*, dengan triso adalah *tri structural isotrophic*. Dalam upaya menguasai teknologi reaktor berjenis HTGR melalui pengembangan RDE, salah tugas yang harus dilaksanakan adalah penguasaan analisis kegagalan bahan bakarnya, khususnya ketika terjadi kecelakaan.

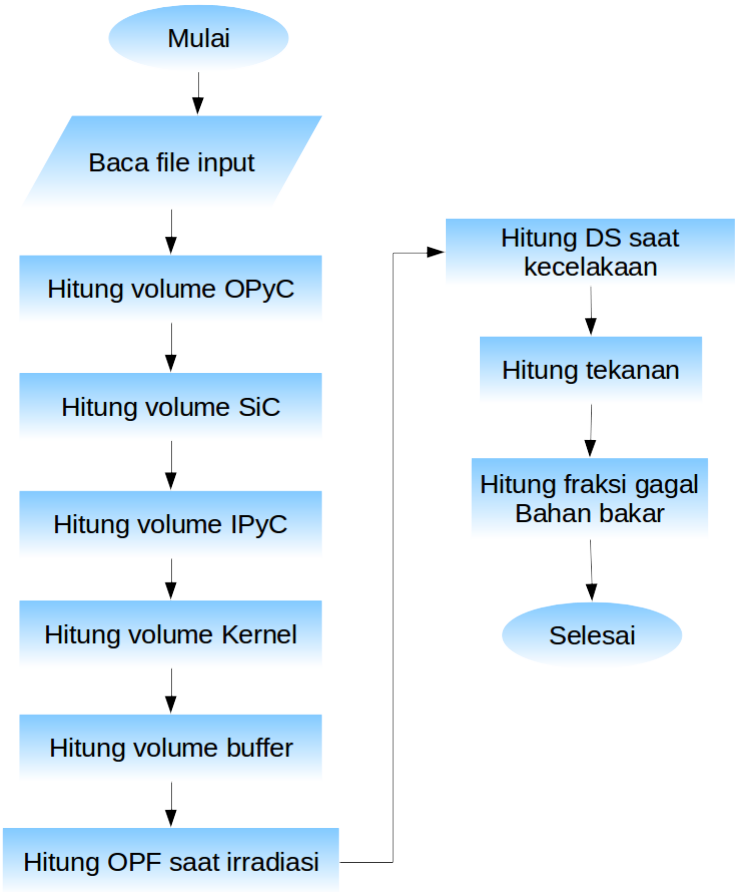
Beragam model analisis telah dikembangkan, salah satunya yang dikembangkan oleh Wang [1]. Selain itu, terdapat sebuah model sederhana yang dikembangkan oleh Verfondern dalam PANAMA [4]. Pada model tersebut, bahan bakar disebut gagal jika kekuatan lapisan SiC (*Silicon Carbide*) lebih kecil daripada tekanan internal dari lapisan di bawahnya. Model inilah yang akan diterapkan dalam TRIAC (*TRIso Analysis Code*).

BAB 2

Alur Perhitungan

2.1 Pendahuluan

Secara umum, perhitungan TRIAC mengikuti diagram alir seperti pada Gambar 2.1 berikut. Penerapannya disajikan dalam Listing 4 berdasarkan pengetahuan yang diperoleh dari dokumen teknis PANAMA [5]. Meski diagram alir tersebut tergambar secara sekuensial, tetapi secara perhitungan ada beberapa formula yang tidak saling tergantung, sehingga urutannya dapat saja dibalik. Hubungan saling ketergantungan antar formula disajikan dalam Gambar 3.1 dan 3.2.



Gambar 2.1: Diagram alir perhitungan TRIAC

2.2 Membaca *file input*

Sub rutin ini ditujukan untuk membaca *file input* dengan format seperti terdapat pada Lampiran 6.2. Sub rutin ini menggunakan skema yang kaku karena identifikasi nilai-nilai yang akan dibaca ditentukan oleh suatu teks tertentu. Setelah teks yang menjadi penanda, nilai-nilai yang dibutuhkan dibaca. Tetapi, nilai tersebut dapat langsung berada dalam satu baris bersama dengan teks penanda, atau berada pada baris yang berbeda. Sub rutin ini terdapat pada Listing 1 dan akan dijelaskan pada sub bab 3.1.

2.3 Menghitung OPF saat irradiasi

OPF (*Oxygen Per Fission*) adalah jumlah atom oksigen yang terlepas selama fisi atom U^{235} atau Pu^{239}. Atom oksigen ini mempengaruhi terbentuknya senyawa CO yang akan meningkatkan tekanan internal dalam bahan bakar. Pembentukan senyawa CO juga dipengaruhi oleh temperatur, waktu serta jenis partikel kernel.

Nilai OPF didekati oleh persamaan (2.1). Nilai n dalam persamaan (2.1) sama dengan banyaknya data sejarah irradiasi. Nilai Δ_i merupakan selisih waktu dari sejarah irradiasi yang dicatat. Nilainya akan berubah dengan berubahnya rentang pencatatan temperatur irradiasi. Jika dalam contoh kasus yang disajikan pada Lampiran 1, rentang waktu pencatatan temperatur irradiasi dilakukan setiap 17 hari, maka Δ_i adalah 17 hari atau $17x24x3600$ detik. t_B adalah waktu irradiasi total bahan bakar, sedangkan $\overline{t_i}$ waktu irradiasi ketika pencatatan dilakukan.

$$OPF \simeq \sum_{i=1}^{n} g(\overline{t_i}) \cdot (t_B - \overline{t_i}) \cdot \Delta t_i \tag{2.1}$$

Tetapi, nilai OPF juga didefinisikan seperti persamaan (2.2), dengan nilai $g(\overline{t_i})$ didefinisikan oleh persamaan (2.3). Nilai R pada persamaan (2.3) adalah konstanta gas sebesar $8.3143[\frac{J}{mole \cdot K}]$.

$$OPF = \frac{g(T)}{2} \cdot t^2 \tag{2.2}$$

$$\frac{g(T)}{2} = 8.32 \cdot 10^{-11} \cdot e^{\frac{-163000}{R \cdot T}} \tag{2.3}$$

Nilai OPF selanjutnya digunakan untuk menghitung nilai temperatur irradiasi (T_B) dari persamaan (2.4). Formula empiris tersebut sesuai untuk jenis bahan bakar UO_2.

$$\log OPF = -10.08 - \frac{0.85 \cdot 10^4}{T_B} + 2 \cdot \log t_B \tag{2.4}$$

Sedangkan nilai T_B akan digunakan untuk menghitung DS, faktor berkurangnya koefisien difusi (s^{-1}) dari gas hasil fisi di dalam partikel kernel. Nilainya untuk bahan bakar UO_2 memenuhi persamaan (2.5).

$$\log DS = -2.30 - \frac{0.8116 \cdot 10^4}{T_B}. \tag{2.5}$$

Terakhir, DS akan digunakan untuk menghitung sebuah nilai tak berdimensi τ_i yang memenuhi persamaan (2.6).

$$\tau_i = DS(T_B) \cdot t_B \tag{2.6}$$

2.4 Menghitung DS saat kecelakaan

Seperti telah dijelaskan dalam sub bab 2.3, *DS* adalah faktor berkurangnya koefisien difusi gas hasil fisi dalam partikel kernel. Sekarang, faktor ini dihitung ketika kondisi kecelakaan terjadi. Kita memerlukan sejarah temperatur bahan bakar setelah kecelakaan terjadi serta τ_i. yang telah dihitung di persamaan (2.6).

Dengan menggunakan persamaan (2.6), kita dapat menghitung nilai *DS* dengan temperatur kecelakaan yang tercatat. Kemudian, kita perlu menghitung nilai τ_A dengan persamaan (2.6) tetapi dengan nilai temperatur dan waktu setelah terjadi kecelakaan. Selanjutnya, dengan modal nilai τ_i dan τ_A kita akan menghitung nilai Fd, yang merupakan faktor fisi gas Xe dan Kr (yang dominan). Nilai Fd dihitung dengan persamaan (2.7).

$$Fd = \frac{(\tau_i + \tau_A) \cdot f(\tau_i + \tau_A) - \tau_A \cdot f(\tau_A)}{\tau_i} \tag{2.7}$$

Sedangkan nilai $f(\tau)$ dihitung menggunakan persaamaan (2.8). Batas atas nilai n pada persaamaan (2.8) dapat menggunakan nilai yang cukup besar, misalnya 1000, atau ketika dua nilai berdekatan yang dihasilkan hanya berselisih kurang dari 10^{-20}. Idealnya, suku penjumlahan sebanyak n akan semakin baik jika hasilnya mendekati 1.

$$f(\tau) = 1 - \frac{6}{\tau} \cdot \sum_{n=1}^{\infty} \left(\frac{1 - e^{-n^2 \cdot \pi^2 \cdot \tau}}{n^4 \cdot \pi^4} \right) \tag{2.8}$$

2.5 Menghitung tekanan

Tekanan adalah variabel yang penting dalam tahapan analisis ini karena akan menentukan fraksi gagal bahan bakar. PANAMA [5] memodelkan fraksi gagal partikel bahan bakar dari sejauh mana lapisan silikon karbida mampu menahan tekanan akibat rilisnya gas produk fisi. Untuk menghitung tekanan yang timbul ketika kecelakaan terjadi pada waktu tertentu, sehingga menyebabkan panas tertentu, digunakan persamaan (2.9) [5].

$$p = \frac{(F_d \cdot F_f + OPF) \cdot F_b \cdot (\frac{V_k}{V_m}) \cdot R \cdot T}{V_f} \tag{2.9}$$

dengan :

F_d = fraksi relatif gas fisi yang lepas

F_f = produk fisi yang dihasilkan dari gas fisi stabil, F_f=0.31

OPF = jumlah atom oksigen setiap terjadi fisi saat terjadi kecelakaan

F_b = *burnup* logam berat (FIMA)

V_f = fraksi void [m^3], terkait dengan 50% volume buffer

V_k = volume kernel [m^3]

V_m = volume molar dalam partikel kernel $\left[\frac{m^3}{mole}\right]$, didefinisikan sebagai rasio berat 1 mol material kernel terhadap kerapatannya. Menurut Verfondern [5], V_m untuk $(Th,U)O_2$, UO_2 dan UCO masing-masing adalah $2.52 \cdot 10^{-5}[\frac{m^3}{mole}]$, $2.44 \cdot 10^{-5}[\frac{m^3}{mole}]$ dan $2.51 \cdot 10^{-5}[\frac{m^3}{mole}]$.

R = konstanta gas, 8.3143 $\left[\frac{J}{(mole \cdot K)}\right]$

Khusus untuk variabel *OPF*, karena perhitungan tekanan dilakukan ketika terjadi kecelakaan, digunakanlah persamaan (2.10). Persamaan (2.10) mirip dengan persamaan (2.4) dengan penambahan suku ke-3.

$$\log OPF = -10.08 - \frac{0.85 \cdot 10^4}{T_B} + 2 \cdot \log t_B - 0.04 \cdot \left(\frac{10^4}{T} + \frac{10^4}{T_B + 75}\right) \tag{2.10}$$

2.6 Fraksi gagal bahan bakar

Tahapan terkahir dari analisis ini adalah perhitungan fraksi gagal bahan bakar. Secara umum, fraksi gagal bahan bakar dipengaruhi sejumlah sebab. Dalam analisis yang dilakukan TRIAC (dan juga PANAMA sebagai acuannya), gagalnya bahan bakar dapat disebabkan oleh 3 sebab. Ketiganya adalah sebagai berikut.

1. Pabrikasi (ϕ_0). Dalam analisis ini, nilai ϕ_0 diasumsikan sama dengan 0.
2. Berkurangnya *tensile strength* lapisan SiC (ϕ_1). Hal ini dapat terjadi karena
   - proses irradiasi maupun
   - meningkatnya temperatur secara signifikan ketika terjadi kecelakaan) atau disebut juga *grain boundary*.
3. Dekomposisi termal pada temperatur tinggi yang menyebabkan terjadinya *weight loss* pada lapisan SiC (ϕ_2).

Ketiga sebab terjadinya kegagalan bahan bakar tersebut mengikuti persamaan (2.11).

$$\phi_{total} = 1 - (1 - \phi_0) \cdot (1 - \phi_1) \cdot (1 - \phi_2) \tag{2.11}$$

2.6.1 Fraksi gagal akibat berkurangnya *tensile strength*

Fraksi gagal partikel triso dimodelkan dengan apa yang diistilahkan Verfondern sebagai model bejana tekan [5]. Hal ini disebabkan karena fraksi gagal dipengaruhi oleh variabel-variabel yang terenkapsulasi dalam parameter tenanan internal dan kekuatan lapisan silikon karbida. Nilai fraksi gagal bahan bakar pada waktu t setelah terjadinya kecelakaan diperoleh dengan persamaan (2.12).

$$\phi_1(t,T) = 1 - e^{-\ln 2 \cdot \left(\frac{\sigma_t}{\sigma_o}\right)^m} \tag{2.12}$$

dengan :

σ_o=*tensile strength* dari SiC [Pa] pada akhir irradiasi

σ_t=tekanan yang dialami SiC [Pa] akibat tekanan gas internal

m=parameter Weibull (dijelaskan selanjutnya)

Variabel tekanan internal pada SiC (σ_t) dihitung dengan dengan persamaan (2.13). Pada persamaan (2.13), jari-jari lapisan SiC merupakan rerata karena lapisan SiC memang memiliki ketebalan yang nilai awalnya diwakili oleh variabel d_o.

$$\sigma_t = \frac{r \cdot p}{2 \cdot d_o} \cdot \left(1 + \frac{\dot{v} \cdot t}{d_o}\right) \tag{2.13}$$

dengan :

r=rerata jari-jari SiC, $\left(0.5 \cdot \left(r_a^3 + r_i^3\right)\right)^{\frac{1}{3}}$ [m]

d_o=ketebalan awal lapisan SiC, $r_a - r_i$ [m]

p=tekanan gas fisi dalam partikel [Pa], dihitung menggunakan persamaan (2.9)

$\dot{v}$=laju korosi sebagai fungsi temperatur (T), $\left[\frac{m}{s}\right]$

Sedangkan variabel laju korosi ($\dot{v}$) dihitung dengan persamaan (2.14), mirip dengan persamaan (2.3) dengan perbedaan pada konstanta.

$$\dot{v} = 5.87 \cdot 10^{-7} \cdot e^{-\left(\frac{179500}{R \cdot T}\right)} \tag{2.14}$$

Selanjutnya, variabel *tensile strength* lapisan SiC, penurunan nilainya mengikuti persamaan (2.15). Variabel σ_{oo} merupakan *tensile strength* awal sebelum diiradiasi. Nilainya merupakan sesuatu yang dapat diukur. Sedangkan Γ dan Γ_s masing-masing merupakan *fluence* netron cepat $\left[10^{25} m^{-2} EDN\right]$ dan *fluence* yang dipengaruhi temperatur irradiasi. Nilai Γ_s ditentukan menggunakan persamaan (2.16).

$$\sigma_o = \sigma_{oo} \cdot \left(1 - \frac{\Gamma}{\Gamma_s}\right) \tag{2.15}$$

$$\log \Gamma_s = 0.556 + \frac{0.065 \cdot 10^4}{T_B} \tag{2.16}$$

Tensile strength lapisan SiC yang dihitung menggunakan persamaan (2.15) merupakan nilai yang berlaku pada satu *coated particle*. Padahal, ada sangat banyak *coated particle* yang dioperasikan. Karena itu, diperlukan perhitungan yang mempertimbangkan variabel ini untuk semua distribusi *coated particles*. Dengan pendekatan yang sama seperti persamaan (2.15), persamaan (2.17) dibangun. Nilai Γ_m ditentukan menggunakan persamaan (2.18).

$$m_o = m_{oo} \cdot \left(1 - \frac{\Gamma}{\Gamma_m}\right) \tag{2.17}$$

$$\log \Gamma_m = 0.394 + \frac{0.065 \cdot 10^4}{T_B} \tag{2.18}$$

Sama seperti σ_{oo}, nilai m_{oo} juga diperoleh dengan mengukur parameter tersebut pada partikel yang belum diiradiasi. Tabel 2.1 menunjukkan nilai σ_{oo} dan m_{oo} pada beberapa jenis specimen sebelum dikenakan irradiasi [5].

Selain korosi karena proses irradiasi, lapisan SiC juga dapat terkorosi karena *grain Boundary*. Jika korosi akibat irradiasi tergantung pada sejarah irradiasi yang dialami bahan bakar dan terjadi sebelum kecelakaan, maka korosi karena *grain Boundary* terjadi setelah kecelakaan. Penurunan nilai distribusi *tensile strength* akibat meningkatnya temperatur karena kecelakaan mengikuti persamaan (2.19), di mana nilai m_o diperoleh dari persamaan (2.17)

$$m = m_o \cdot \left(0.44 + 0.56 \cdot e^{-\dot{\eta} \cdot t}\right) \tag{2.19}$$

dan nilai $\dot{\eta}$ mengikuti persamaan 2.20 dengan pola yang sama seperti persamaan (2.14).

$$\dot{\eta} = 0.565 \cdot e^{\left(\frac{-187400}{R \cdot T}\right)} [s^{-1}] \tag{2.20}$$

Tabel 2.1: Nilai σ_{oo} dan m_{oo} untuk berbagai jenis specimen[5]

Specimen	Sebelum irradiasi		Setelah irradiasi	
	σ_{oo} [MPa]	m_{oo}	σ_o [MPa]	m_o
EO 1674	722	7.0	660	6.1
EO 1607	850	8.0	777	7.0
HT 150-167	600	6.0	549	5.3
EO 249-251	453	5.0	414	4.4
EO 403-405	867	8.4	793	7.4
EUO 1551	1060	8.5	969	7.4
ECO 1541	1080	6.4	987	5.6
EC 1338	998	7.4	912	6.5

2.6.2 Fraksi gagal bahan bakar akibat *weight loss*

Laju *weight loss* yang terjadi akibat tingginya temperatur saat terjadi kecelakaan mengikuti persamaan (2.21).

$$k = k_o \cdot e^{\frac{-Q}{R \cdot T}} \tag{2.21}$$

dengan $Q = 556 \left[\frac{kJ}{mol}\right]$ dan k_o adalah faktor frekuensi yang tergantung pada jenis partikel.

Selanjutnya, diasumsikan bahwa partikel TRISO tergantung pada apa yang disebut sebagai "*action integral*", dan disimbolkan dengan ζ yang nilainya mengikuti persamaan (2.22).

$$\zeta = \int_{t_1}^{t_2} k(T)dt \tag{2.22}$$

dengan $K(T)$ adalah nilai yang menggambarkan sejarah kondisi partikel yang bergantung pada temperatur dan waktu.

Secara numerik, persamaan (2.22) dapat dituliskan sebagai persamaa (2.23).

$$\zeta(t_2) = \zeta(t_1) + k(T_m) \cdot (t_2 - t_1) \tag{2.23}$$

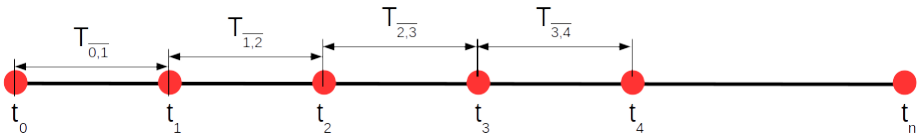
dengan $k(T_m) = \frac{375}{d_o} \cdot e^{\left(\frac{-556000}{R \cdot T_m}\right)}$.

Kemudian, fraksi gagal ϕ_2 sedemikian rupa sehingga nilainya ≤ 1. Karena itu, variabel ϕ_2 selanjutnya didefinisikan sebagai persamaan (2.24).

$$\phi_2(t,T) = 1 - e^{-\alpha \cdot \zeta^{\beta}} \tag{2.24}$$

Nilai α dan β kemudian ditentukan secara empiris. Dan berdasarkan penelitian empiris sebelumnya terhadap partikel UO_2, diperoleh nilai $\alpha = \ln 2 = 0.693$, sedangkan nilai $\beta = 0.88$.

Dalam TRIAC, faktor fraksi gagal ini tidak akan dipertimbangkan. Hal ini disebabkan karena kondisi ini terjadi pada temperatur di atas 2000 °C. Sementara RDE tidak dirancang untuk sampai pada temperatur tersebut.



Gambar 2.2: Hubungan antara waktu dan temperatur pada perhitungan ϕ_1

2.6.3 Pertumbuhan fraksi gagal

Berdasarkan PANAMA [5], Verfondern memodelkan pertumbuhan fraksi gagal partikel triso akibat berkurangnya *tensile strength* adalah seperti persamaan (2.25) berikut.

$$\phi_1 = \phi_1(t_2, T_m) - \phi_1(t_1, T_m) \tag{2.25}$$

T_m merupakan temperatur rata-rata antara waktu t_1 dan t_2. Ilustrasinya disajikan dalam Gambar 2.2

Disebutkan Verfondern [5], nilai ϕ_1 saat kecelakaan dimulai (t_0 seperti pada Gambar 2.2) merupakan fungsi dari sejarah irradiasi. Sedangkan untuk waktu-waktu selanjutnya $(t_1, t_2, \cdots, t_n)$ merupakan akumulasi dari nilai ϕ_1 pada persamaan (2.25). Jika ϕ_1 pada t_1 lebih besar daripada ϕ_1 pada saat t_0, maka akumulasikan nilai ϕ_1. Jika sebaliknya, gunakan nilai ϕ_1 sebelumnya untuk perhitungan selanjutnya (nilai ϕ_1 tetap).

Sebagai ilustrasi, saat menghitung nilai ϕ_1 di $t = t_1$, maka diperlukan nilai $\phi_1(t_0, T_m)$ (nilai pertama) dan $\phi_1(t_1, T_m)$ (nilai kedua). Nilai pertama adalah fungsi irradiasi, sedangkan nilai kedua diperoleh dari persamaan (2.12) dengan parameter-parameter yang sesuai. Selisih keduanya akan menentukan nilai ϕ_1 di titik $t = t_1$. Jika selisih nilai kedua dan pertama positif, selisih nilai tersebut diakumulasikan pada nilai ϕ_1 di $t = t_0$. Tetapi jika sebaliknya, maka nilai ϕ_1 di titik $t = t_1$ sama dengan nilai ϕ_1 di titik $t = t_0$. Skenario yang sama berlaku untuk titik-titik waktu selanjutnya.

2.7 Modul *sampling*

2.7.1 Pendahuluan

Latin Hypercube Sampling (LHS) saat ini telah menjadi metode sampling yang paling banyak digunakan dalam analisis keandalan dan ketidakpastian pada analisis sistem kompleks berbasis Monte-Carlo [6, 7]. LHS sendiri didefinisikan sebagai metode untuk menghasilkan *sample* acak dari nilai parameter [8]. Alasan LHS banyak digunakan dalam metode berbasis Monte Carlo adalah karena kemampuannya mengurangi jumlah eksekusi untuk memperoleh hasil yang cukup akurat.

LHS dapat dimodelkan sebagai sebuah fungsi $y = f(x)$, dengan f merepresentasikan model dari sistem yang sedang dikaji, $x = [x_1, x_2, \ldots]$ merupakan vektor input bagi model, dan $y = [y_1, y_2, \ldots]$ merupakan vektor prediksi model [6]. Tujuan dari analisis ketidakpastian ini adalah untuk menentukan ketidakpastian elemen y sebagai akibat dari ketidakpastian elemen x. Dalam konteks TRIAC, LHS dapat digunakan untuk menentukan ketidakpastian fraksi gagal bahan bakar triso sebagai akibat ketidakpastian dimensi partikel tersebut. Tabel 2.2 menunjukkan contoh ketidakpastian parameter dalam partikel triso [9]

TRIAC selanjutnya juga akan dilengkapi dengan SRS (Simple Random Sampling). LHS dan SRS selanjutnya digabung menjadi modul *sampling* pada TRIAC.

Tabel 2.2: Nilai nominal dan variasinya pada komposisi partikel triso[9]

Item	Nilai Nominal	Toleransi disain	Rentang teramati	Standar deviasi
Uranium fuel loading ($\frac{g}{fuel\ pebble}$)	$5.0g$	$5.0 \pm 0.1g$	$4.95 - 5.05g$	n/a
Density of graphite in matrix and outer shell of fuel pebble	$1.73 \frac{g}{cm^3}$	$1.75 \pm 0.02 \frac{g}{cm^3}$	$1.73 \frac{g}{cm^3}$	n/a
Total ash in fuel element	0	$\leq 300.0\ ppm$	$130 - 190\ ppm$	n/a
Lithium in fuel element	0	$\leq 0.3\ ppm$	$0.007 - 0.023\ ppm$	n/a
Boron in fuel element	1.3 ppm	$\leq 0.3\ ppm$	$0.15\ ppm$	n/a
Ratio of oxygen to uranium in kernel	2	< 2.01	n/a	n/a
Density of kernel	$10.4 \frac{g}{cm^3}$	$> 10.4 \frac{g}{cm^3}$	$10.83 \frac{g}{cm^3}$	n/a
Density of buffer layer	$1.1 \frac{g}{cm^3}$	$\leq 1.1 \frac{g}{cm^3}$	$1.02 \frac{g}{cm^3}$	teramati $0.03 \frac{g}{cm^3}$
Density of IPyC layer	$1.9 \frac{g}{cm^3}$	$1.1 \pm 0.1 \frac{g}{cm^3}$	$1.86 \pm 0.06 \frac{g}{cm^3}$	n/a
Density of SiC layer	$3.18 \frac{g}{cm^3}$	$\geq 3.18 \frac{g}{cm^3}$	$3.21 \pm 0.02 \frac{g}{cm^3}$	n/a
Density of OPyC layer	$1.9 \frac{g}{cm^3}$	$1.9 \pm 0.1 \frac{g}{cm^3}$	$1.87 \pm 0.02 \frac{g}{cm^3}$	n/a
Density of reflector graphite	$1.6 \frac{g}{cm^3}$	n/a	n/a	n/a

2.7.2 Opsi distribusi

LHS bekerja dengan tahapan sebagai berikut [10].

1. Mendefinisikan variabel $Y = f(x_i)$. Pendefinisian variabel Y melibatkan
   - jumlah variabel X
   - konstanta setiap variabel x_i
2. Mendefinisikan variabel X. Beberapa tahapan yang perlu dilakukan adalah sebagai berikut.
   - Menentukan jenis distribusi. Untuk TRIAC, jenis distribusi yang akan digunakan adalah normal, *uniform* dan triangular. Sample yang dihasilkan jumlahnya akan mengikuti distribusi yang ditetapkan.
     - Distribusi normal membutuhkan parameter rerata (μ) dan varian (σ)
     - Distribusi *uniform* membutuhkan parameter n_1 dan n_2, dengan n_1 dan n_2 masing-masing adalah nilai terendah dan tertinggi
     - Distribusi triangular membutuhkan parameter *min*, *mod*, *max*, yang masing-masing adalah nilai terendah, nilai yang paling sering muncul dan nilai tertinggi.
   - Menghasilkan *sample* untuk variabel X dengan tahapan sebagai berikut.
     - Bangkitkan bilangan random (r) dalam rentang 0 dan 1.
     - Hitung nilai P_m menggunakan persamaan (2.26) sebagai instrumen untuk memastikan semua *sample* (sejumlah N) yang dihasilkan tercuplik dari setiap *mesh region* (m).
     - Menghitung nilai X dengan persamaan (2.27). Fungsi F^{-1} tergantung dari distribusi yang menjadi target dan akan dijelaskan lebih detil dalam sub bab 2.7.3.

$$P_m = \frac{1}{N} \cdot (r + m - 1) \tag{2.26}$$

$$X = F^{-1}(P_m) \tag{2.27}$$

3. Setelah sample untuk variabel X diperoleh, variabel Y dapat dihitung. Nilai Y sesuai nilai setiap sample X yang ada selanjutnya dirata-rata dan dihitung variannya (σ). Akan tetapi, untuk SRS, fungsi P_m tidak digunakan. Nilai random (r) yang dihasilkan langsung menjadi masukan bagi fungsi inverse untuk mendapatkan nilai X.

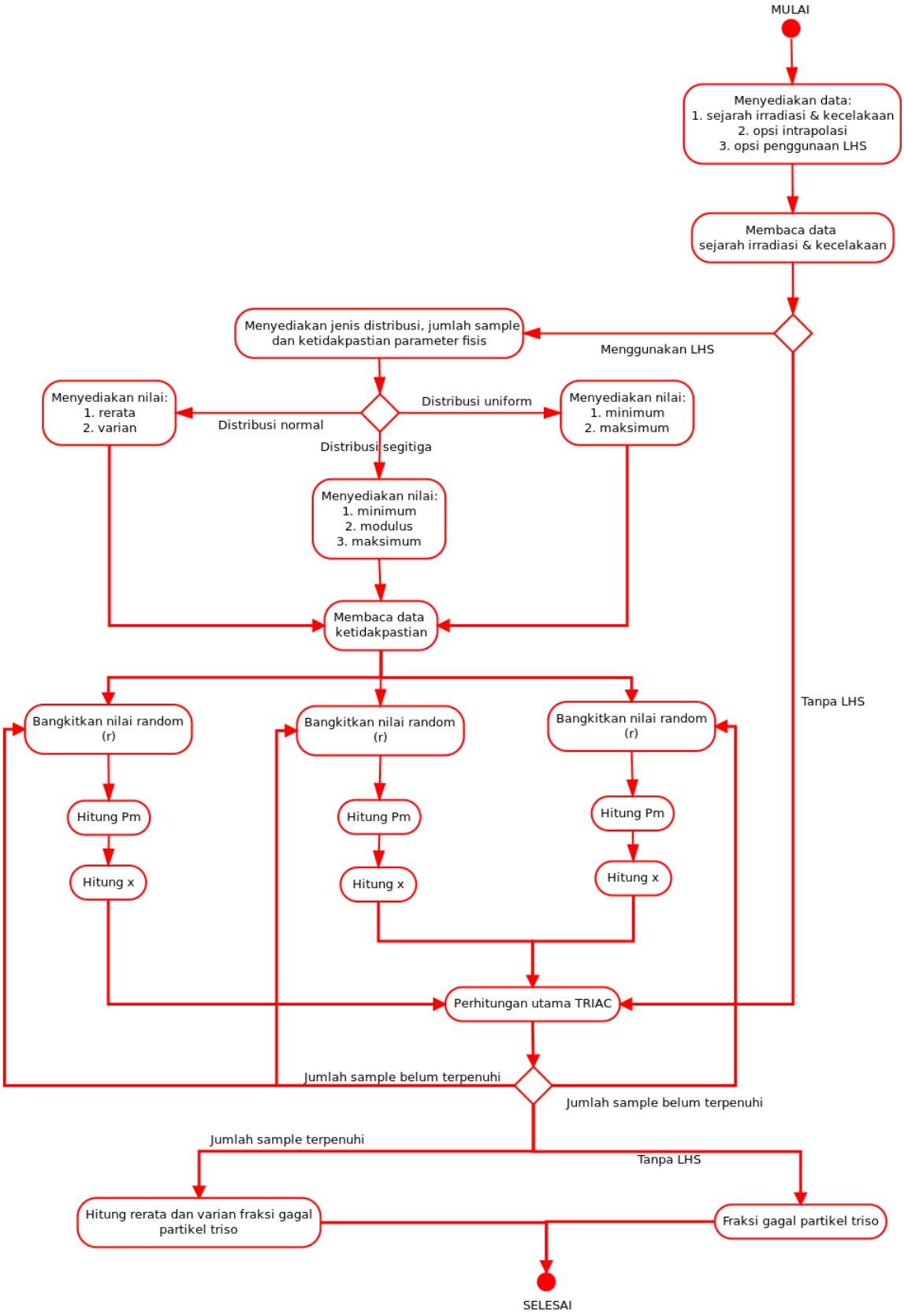
2.7.3 Fungsi *inverse* untuk mendapatkan nilai X

Berikut adalah fungsi inverse (F^{-1}) untuk distribusi dan kondisinya.

1. Distribusi normal $\rightarrow X = n_2 \cdot \sqrt{2} \cdot ((2 \cdot P_m) + n_1)$
2. Distribusi *uniform* $\rightarrow X = (P_m \cdot (n_2 - n_1)) + n_1$
3. Distribusi triangular
   - kondisi $P_m \leq k \rightarrow X = n_1 + \sqrt{P_m \cdot (n_3 - n_1) \cdot (n_2 - n_1)}$
   - kondisi $P_m > k \rightarrow X = n_3 + \sqrt{(1 - P_m) \cdot (n_3 - n_1) \cdot (n_3 - n_2)}$
   - dengan kondisi k adalah
     - $k = 0.0$ jika $n_1 = n_2$
     - $k = 1.0$ jika $n_2 = n_3$
     - $k = \frac{n_2 - n_1}{n_3 - n_1}$ jika $n_1 \neq n_2$ dan $n_2 \neq n_3$

2.7.4 Alur eksekusi

Ketika LHS digunakan, perhitungan yang telah dijelaskan sebelumnya akan diulang sebanyak *sample* yang dibutuhkan. Karena itu, ketika opsi LHS digunakan, *looping* terluar adalah *looping* LHS sebanyak *sample*. Secara umum, eksekusi TRIAC akan mengikuti alur seperti diagram aktifitas di Gambar 2.3.



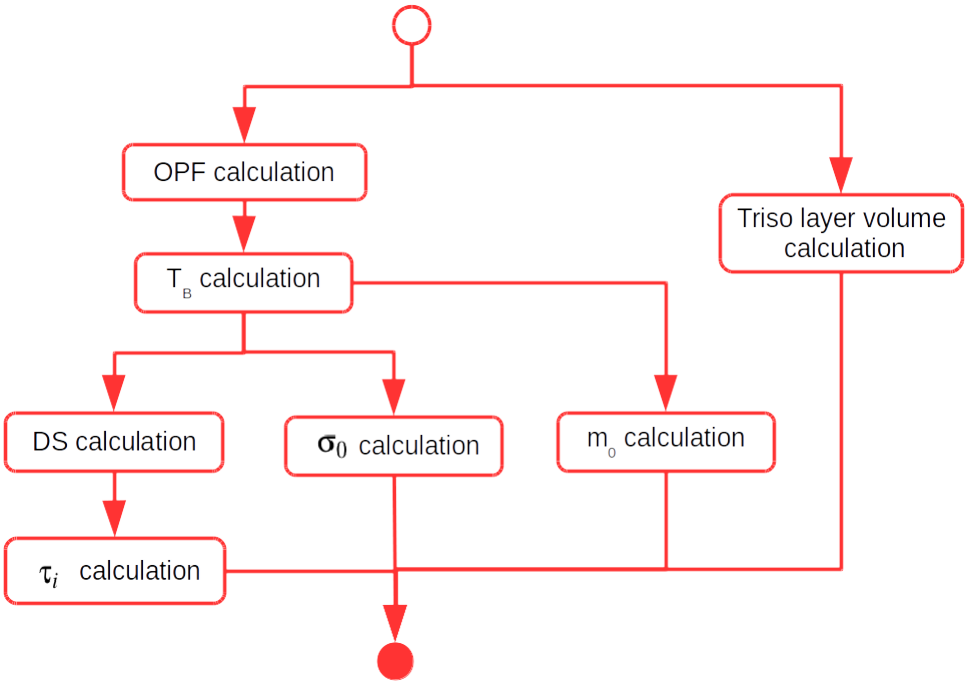
Gambar 2.3: Diagram aktifitas eksekusi TRIAC

BAB 3

Penerapan TRIAC

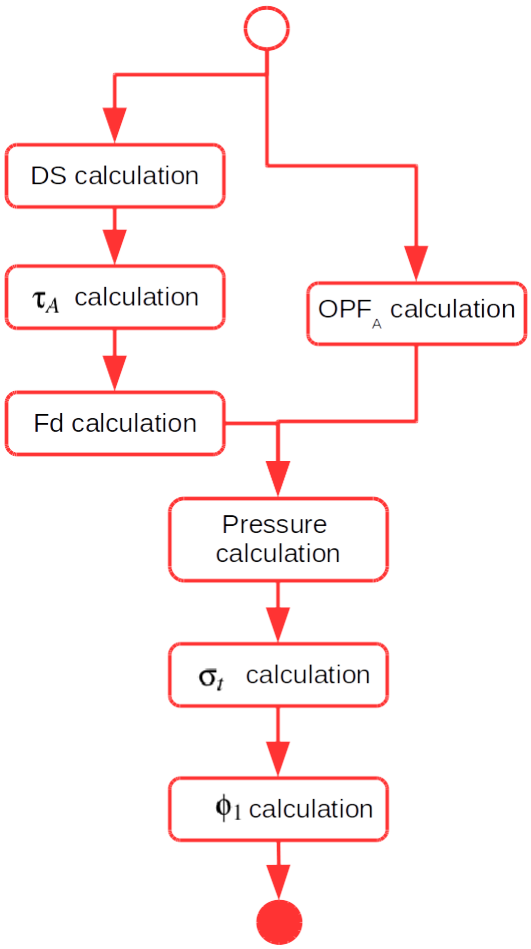
3.1 Pendahuluan

TRIA *Code* yang telah dijelaskan sebelumnya secara umum dapat dikelompokkan menjadi dua tugas utama, masing-masing adalah perhitungan di waktu irradiasi dan kecelakaan. Saat irradiasi, hubungan saling ketergantungan antar variabel adalah seperti Gambar 3.1. Sedangkan saat kecelakaan, hubungannya adalah seperti pada Gambar 3.2.



Gambar 3.1: Hubungan ketergantungan antar variabel di fase irradiasi

Selanjutnya, triac juga memerlukan sejumlah data yang harus diberikan oleh pengguna sebelum perhitungan dimulai. Selain data-data seperti yang akan dijelaskan dalam sub bab 3.2, diperlukan juga beberapa data lain. Karena triac mengadopsi perhitungan yang dilakukan dalam PANAMA [5], maka triac juga memerlukan data seperti yang diperlukan PANAMA. Tabel 3.1 menyajikan beberapa parameter serta nilainya yang diperlukan oleh triac, masing untuk HTR-Modul dan HTR-500.



Gambar 3.2: Hubungan ketergantungan antar variabel di fase kecelakaan

Selain itu, triac juga memerlukan parameter lain berupa status interpolasi. Dengan status ini, sejarah irradiasi/kecelakaan akan diinterpolasi atau menggunakan nilai yang diberikan pengguna dari *file input*.

3.2 Pembacaan *file input*

Seluruh proses dalam triac didahului dengan membaca *file input* dengan format yang sama seperti pada Lampiran 6.2. Penerapan pembacaan *file input* adalah seperti pada Listing 1.

Di Listing 1, pembacaan *input data* dilakukan secara sekuensial dan manual. Nilai-nilai yang harus dibaca ditentukan berdasarkan informasi yang ada pada *file input*. Sebagai contoh, untuk membaca nilai geometri, digunakan karakter ”[m]” sebagai penanda. Jika ditemukan karakter tersebut, maka di saat itulah pembacaan nilai geometri dilakukan. Hal inilah yang dimaksud sebagai pembacaan secara manual. Ketika karakter yang diperlukan berubah, maka modifikasi harus dilakukan pada modul ini.

Selain nilai terkait geometri, diperlukan juga pembacaan untuk nilai *physical properties* serta sejarah operasi, baik saat operasi normal maupun kecelakaan. Pembacaan nilai yang berbeda dilakukan secara berurutan berdasarkan kemunculan nilai tersebut dalam *file input*. Hal inilah yang dimaksud dengan pembacaan secara sekuensial.

Tabel 3.1: Tambahan data yang diperlukan triac[5]

Parameter	HTR-Modul	HTR-500
Jenis partikel	UO_2	UO_2
Burnup / FIMA / Fb	0.08	0.08
Fast fluence / Γ $[10^{25} m^{-2}]$	1.4	1.4
σ_{oo} [MPa]	834	834
m_{oo}	8.02	8.02

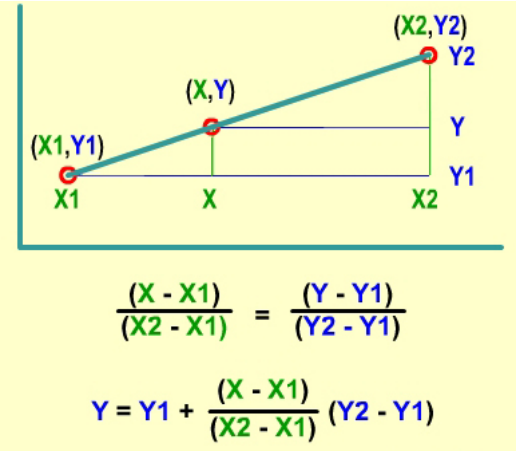
Terdapat empat jenis data yang perlu dibaca dari *file input* dalam Lampiran 1, masing-masing adalah sebagai berikut. Penerapannya disajikan dalam Listing 1.

1. Data tentang geometri *pebble*. Data ini diidentifikasi menggunakan teks yang didefinisikan oleh variabel `statusGeometry` (baris ke-4. Di dalam data geometri, terdapat empat data berbeda, masing-masing secara berurutan adalah panjang jejari *pebble* terluar, OPyC (*Outer Pyrolitic Carbon*), SiC (*Silicon Carbide*), IPyC (*Inner Pyrolitic Carbon*), *buffer* dan kernel. Data geometri akan digunakan untuk menghitung volume setiap elemen pelapis (Gambar 1.1). Yang perlu diperhatikan adalah data jari-jari yang disajikan adalah jarak dari pusat bahan bakar sampai titik terluar dari setiap lapisan. Karena itu, volume suatu lapisan harus mempertimbangkan lapisan-lapisan di dalamnya. Data geometri disimpan dalam variabel diberi nama `dimensi` dan dalam bentuk `list` (baris ke-9).

2. Data tentang kekuatan SiC. Data ini diidentifikasi menggunakan teks yang didefinisikan oleh variabel `statusCharacteristics` (baris ke-5 pada Listing 1). Ada empat nilai yang perlu dibaca terkait kekuatan SiC, masing-masing adalah SiC *Tensile Strength* [Pa], *Weibull Modulus Burnup* [FIMA], *Fission Yield of stable fission gasses* [Ff], *Fast Neutron Fluence* dan rasio berat Th terhadap U-235 pada kernel. Data terkait kekuatan SiC disimpan dalam variabel yang diberi nama `characteristics` dalam bentuk `list` (baris ke-10).

3. Data tentang sejarah irradiasi. Data ini diidentifikasi menggunakan teks yang didefinisikan oleh variabel `statusIrradiation` (baris ke-6 pada Listing 1). Data ini merupakan data temperatur bahan bakar *pebble* pada selang waktu tertentu. Sebagai contoh, data yang disajikan pada Lampiran 1 diambil pada selang waktu 17 hari. Data sejarah irradiasi disimpan dalam variabel yang diberi nama `irradiation` dalam bentuk `list`. Setiap elemen adalah `list` yang secara *nested* terdiri dari dua elemen yang mewakili data kolom kedua dan ketiga tiap akuisisi (baris ke-11). Ilustrasinya adalah seperti $[[0,593],[1468800,833],\ldots]$ dengan informasi waktu pengukuran dalam satuan detik. Data tentang nomor urut tidak digunakan karena selain tidak diperlukan dalam perhitungan, akan menyulitkan proses interpolasi yang akan diterapkan berikutnya.

4. Data tentang sejarah kecelakaan. Data ini diidentifikasi menggunakan teks yang didefinisikan oleh variabel `statusAccident` (baris ke-7). Data ini memiliki pola yang sama dengan data sejarah irradiasi. Data sejarah keselakaan disimpan dengan cara yang sama seperti data tentang sejarah irradiasi tetapi dengan nama `accident` (baris



Gambar 3.3: Ilustrasi interpolasi linier yang digunakan

ke-12). Ilustrasinya adalah seperti $[[0, 1033], [2341.44, 1033], \ldots]$ dengan informasi waktu pengukuran dalam satuan detik.

.

Namun, terlihat pada baris ke-76 dari Listing 1, terdapat total 5 variabel yang dikembalikan ke fungsi awal, dengan variabel kelima adalah $b - a$. Variabel ini adalah rentang waktu pengukuran data irradiasi.

Selain itu, untuk meningkatkan ketelitian perhitungan, disiapkan juga modul interpolasi secara linier. Modul ini disiapkan agar sejarah operasi normal dan kecelakaan sehingga dapat diperoleh hasil yang tepat. Penerapan dari modul interpolasi linier tersebut disajikan pada Listing 2.

Seperti terlihat pada Lampiran 6.2, sejarah operasi normal atau disebut juga sebagai sejarah irradiasi, terdapat 3 kolom dalam *file input*. Demikian juga untuk sejarah ketika terjadi kecelakaan. Ketiganya adalah nomor urut, hari ke sekian dan temperatur. Dengan melakukan interpolasi, selisih hari yang digunakan dapat diperkecil. Dalam contoh *file input*, selisih pencatatan adalah 17 hari. Dengan interpolasi, kita dapat mengestimasi sejarah dalam selisih waktu yang lebih singkat.

Interpolasi yang diterapkan dapat diilustrasikan dalam Gambar 3.3 [1]. Argumen ketiga dari fungsi `linier (a,b,c)`, c, adalah jumlah partisi diantara nilai x_1 dan x_2. Nilai tersebut adalah *dt* yang merupakan argumen ketika mengeksekusi kode komputer TRIAC (Listing 4). Penggunaan fungsi interpolasi ini dilakukan di Listing 4 pada baris ke-53 s/d 66.

3.3 TRIAC Core

Modul ini adalah modul yang diterapkan dalam bentuk *class* seperti terlihat pada Listing 3. Modul ini dipersiapkan menjadi modul utama dalam TRIAC, baik ketika melibatkan perhitungan LHS maupun tidak.

Modul untuk perhitungan TRIAC dibuat dalam bentuk class yang menjalankan sejumlah fungsi dan menyimpan beberapa parameter dan konstanta yang diperlukan dalam perhitungan. Konstanta dan parameter diinisiasi dalam fungsi `__init__` serta dilengkapi sejumlah fungsi untuk memodifikasi nilainya. Berikut adalah konstanta dan parameter yang terlibat. Dari daftar tersebut, hanya `R` yang tidak memiliki fungsi modifikasi. Sedangkan

[1] `http://jadipaham.com/wp-content/uploads/2015/10/Rumus-interpolasi-linear.jpg`

`r` dan `d0` dimodifikasi melalui fungsi yang sama karena keduanya membutuhkan argumen yang sama. Yang selanjutnya masuk dalam kategori parameter adalah waktu (tb) dan temperatur irradiasi (Tb). Hal ini disebabkan karena parameter tersebut dihitung satu kali untuk kemudian digunakan dalam perhitungan fraksi gagal partikel triso di setiap *mesh* waktu kecelakaan.

1. `R`, konstanta gas, 8.3143 $\left[\frac{J}{mole \cdot K}\right]$
2. `Ff`, produk fisi yang dihasilkan dari gas fisi stabil
3. `Fb`, *burnup* logam berat / FIMA
4. `Vm`, volume molar dalam partikel kernel yang dipengaruhi oleh material kernel, $\left[\frac{m^3}{mole}\right]$
5. `m0`, parameter weibull bahan bakar sebelum digunakan
6. `r` (rerata jari-jari lapisan SiC) dan `d0` (ketebalan awal lapisan SiC)
7. `sigma0`, *tensile strength* SiC di akhir irradiasi [Pa]
8. `Vk`, volume kernel [m^3]
9. `Vf`, fraksi void, 0.5 volume buffer.
10. `tb`, waktu irradiasi, detik
11. `Tb`, temperatur rerata irradiasi, ^oC
12. Γ, fluence neutron cepat, $10^{25} m^{-2}$ EDN

Sedangkan daftar fungsi yang terdapat dalam class `Core` adalah sebagai berikut.

1. Perhitungan volume lapiran triso.
   - Nama fungsi: `volume`
   - Argumen: `radius` (jari-jari lapisan partikel triso)
   - Formula yang dikerjakan: $v = \frac{4}{3}\,\pi\, radius^2$
   - *Return value*: `v`
2. OPF (*Oxygen Per Fission*)
   - Nama fungsi: `OPF`
   - Argumen: `irradiation,y`
     - `irradiation`: *array* berdimensi dua, dengan kolom pertama adalah waktu dengan satuan detik sedangkan kolom kedua adalah temperatur dengan satuan ^oC
     - `y`: rentang waktu pengukuran sejarah irradiasi dengan satuan detik. Dikaitkan dengan argumen `irradiation`, maka `y` adalah rentang waktu antara dua waktu berurutan di dalamnya.
   - Formula yang dikerjakan: Listing 3.1
   - *Return value*: `z`

Listing 3.1: Fungsi OPF

```
def OPF(self,irradiation,y):
        x=len(irradiation)
        print("Irradiation length:",x)
        z=0.0
        for i in range(x):
                j=irradiation[i]
                a1=self.R*j[1]
                a=-163000/(a1)
                b=math.exp(a)
                g=2*(8.32e-11)*b
                g1=g*(self.tb-j[0])*y
                z=z+g1
  return z
```

3. F_τ

   - Nama fungsi: `FTau`
   - Argumen: `tau`, salah satu parameter dalam sejumlah formula empiris pada PANAMA [5]
   - Formula yang dikerjakan: Listing 3.2
   - *Return value*: `ftau`

Listing 3.2: Fungsi FTau

```
def FTau(self,tau):
        looping=0.0
        for n in range (1,2000):
                pangkat=math.pow(n,2)*math.pow(math.pi,2)*tau
                A=math.exp(-(pangkat))
                B=math.pow(n,4)*math.pow(math.pi,4)
                looping=looping+((1-A)/B)
        ftau=1-((6/tau)*looping)
        return ftau
```

4. DS

   - Nama fungsi: `DS`
   - Argumen: `T`, temperatur dalam ^oC
   - Formula yang dikerjakan: Listing 3.3
   - *Return value*: `ds`

Listing 3.3: Fungsi DS

```
def DS(self,T):
        logDS=-2.3-(8116/T)
        ds=math.pow(10,logDS)
        return ds
```

5. OPF Accident

   - Nama fungsi: `OPFAccident`
   - Argumen:
     - `T`: temperatur ketika terjadi kecelakaan, ^oC
   - Formula yang dikerjakan: Listing 3.4
   - *Return value*: `opfa`

Listing 3.4: Fungsi OPFAccident

```
def OPFAccident(self,T):
        logOPF=-10.08-(8500/self.Tb)+(2*math.log10(self.tb))-
        (0.404*((10000/T)-(10000/(self.Tb+75))))
        opfa=math.pow(10,logOPF)
        return opfa
```

6. Parameter weibull

   - Nama fungsi: `weibullParam`
   - Argumen:
     - `T`: temperatur ketika terjadi kecelakaan, ^oC
     - `t`: waktu ketika terjadi kecelakaan, detik
   - Formula yang dikerjakan: Listing 3.5
   - *Return value*: `m`
   - Catatan: pada penerapannya, perhitungan ini tidak dilakukan. Jika dilibatkan dalam perhitungan, maka fraksi gagal partikel triso akan lebih besar daripada hasil yang diperoleh PANAMA. Meskipun hal itu logis, karena temperatur yang lebih tinggi akan meningkatkan potensi kerusakan partikel triso, tetapi tahap awal pengembangan TRIAC mentargetkan sedekat mungkin hasil yang diperoleh dengan PANAMA. Karena itu, diasumsikan bahwa parameter weibull tidak berubah selama kecelakaan terjadi. Seperti terlihat pada Listing 3, fungsi untuk menghitung parameter weibull terkini tidak dijalankan, dan hanya bertugas mengembalikan nilai parameter, tepat saat dimulainya kecelakaan (m_0).

Listing 3.5: Fungsi weibullParam

```
def weibullParam(self,T,t):
        logGammaM=0.394+(650/self.Tb)
        gammaM=math.pow(10,logGammaM)
        m0=self.m00*(1-(self.Fluence/gammaM))
        a=-187400/(self.R*T)
        b=math.pow(math.e,a)
        etaDot=0.565*b
        c=math.pow(math.e,-etaDot*t)
        m=m0*(0.44+(0.56*c))
        return m
```

7. tekanan

   - Nama fungsi: `tekanan`
   - Argumen:
     - `Fd`: fraksi gas fisi yang lepas
     - `opf`: *oxygen per fission* saat terjadi kecelakaan
     - `T`: temperatur ketika terjadi kecelakaan, ^oC
   - Formula yang dikerjakan: Listing 3.6
   - *Return value*: `p`

Listing 3.6: Fungsi tekanan

```
def tekanan(self,Fd,opf,T):
        p=((Fd*self.Ff)+opf)*self.Fb*(self.Vk/self.Vm)*self.R*T/self.Vf
        return p
```

8. σ_T

- Nama fungsi: `sigmaT`
- Argumen:
  - `p`: tekanan (Pa)
  - `t`: waktu terjadi kecelakaan, detik
  - `T`: temperatur ketika terjadi kecelakaan, ^oC
- Formula yang dikerjakan: Listing 3.7
- *Return value*: `a`

Listing 3.7: Fungsi *tensile strength* pada temperatur T

```
1        def SIGMA_T(self,p,t,T):
2                nu=(5.87e-7)*math.exp(-179500/(self.R*T))
3                a=(self.r*p/(2*self.d0))*(1+(nu*t/self.d0))
4                return a
```

9. ϕ

- Nama fungsi: `phi`
- Argumen:
  - `sigmaT`: *tensile strength* pada temperatur `T`
  - `m`: parameter weibull
- Formula yang dikerjakan: Listing 3.8
- *Return value*: `d`

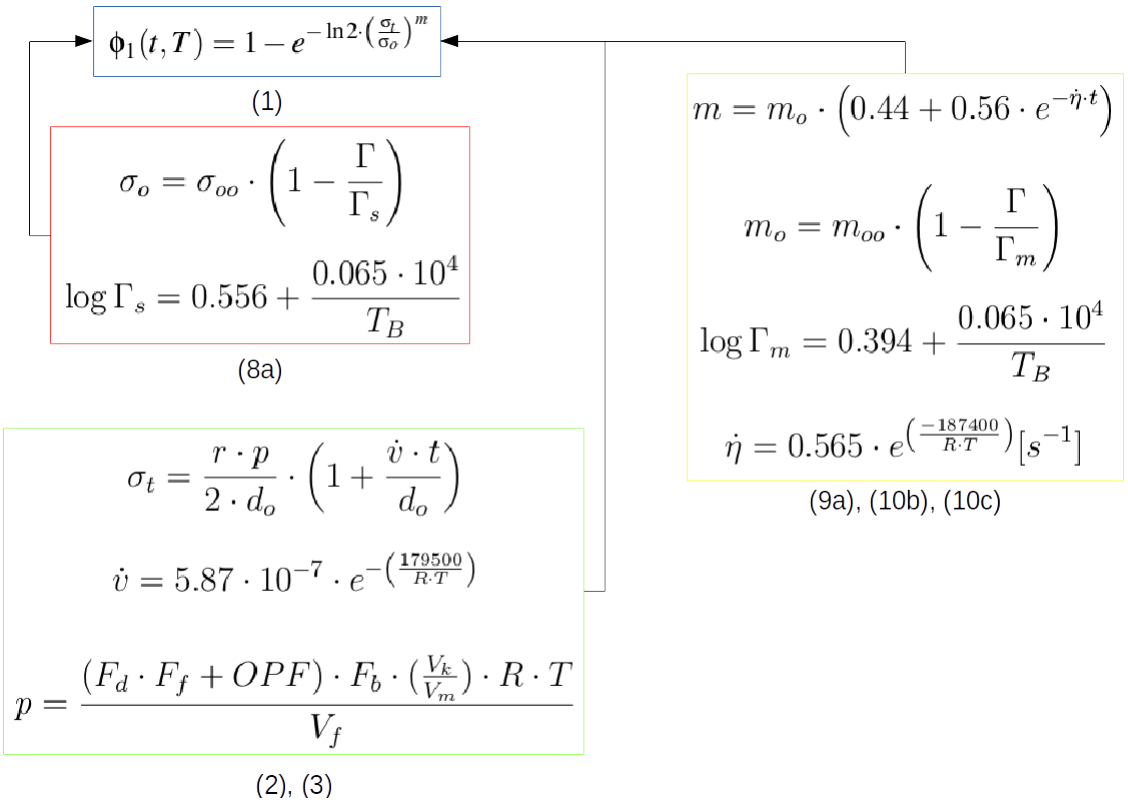
Listing 3.8: Fungsi fraksi gagal partikel triso

```
1        def PHI(self,sigmaT, m):
2                a=sigmaT/self.sigma0
3                b=math.pow(a,m)
4                c=math.log(2)*b
5                d=1-math.exp(-c)
6                return d
```

3.4 Perhitungan TRIAC

Bagian ini adalah bagian pengendali dari perhitungan TRIAC yang alur eksekusinya diilustrasikan pada Gambar 2.1. Sedangkan hubungan interaksi antar fungsi untuk mendapatkan nilai fraksi gagal partikel triso setelah sekian waktu sejak terjadi kecelakaan dapat diilustrasikan seperti Gambar 3.4. Kotak dengan warna merah, kuning dan hijau pada Gambar 3.4 menunjukkan formula-formula yang hasilnya menjadi masukan untuk formula pada kotak berwarna biru. Sementara angka di bawah kotak-kotak tersebut adalah nomor formula dalam dokumen PANAMA [5].

Sedangkan Gambar 3.5 merupakan kelanjutan dari interaksi yang ditunjukkan Gambar 3.4, khususnya untuk menyediakan nilai masukan bagi parameter nilai tekanan yang dialami lapisan silikon karbida. Sama seperti Gambar 3.4, angka di bawah kotak-kotak berwarna yang berisi formula pada Gambar 3.5 menunjukkan nomor formula pada dokumen PANAMA [5]. Selain informasi nomor persamaan pada dokumen PANAMA, ditunjukkan pula bahwa kotak berwarna kuning merupakan variabel yang dipengaruhi oleh jenis partikel triso. Formula yang disajikan merupakan formula empiris untuk jenis partikel UO_2. Sementara untuk kotak berwarna hijau, selain dipengaruhi oleh jenis material partikel triso, juga dipengaruhi oleh kondisi apakah partikel triso sedang berada pada masa irradiasi (formula pertama) atau kecelakaan (formula kedua).



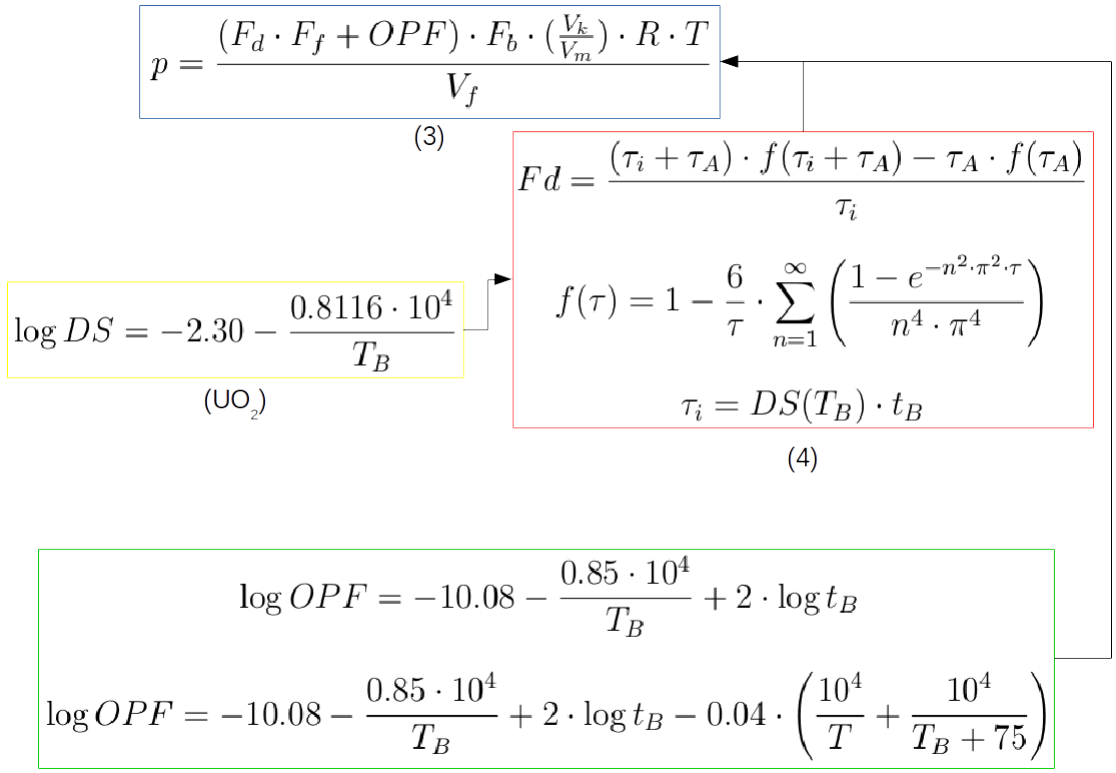
Gambar 3.4: Interaksi antar fungsi untuk mendapatkan fraksi gagal partikel triso

Program ini dirancang untuk dapat menerima tiga kombinasi argumen, masing-masing adalah eksekusi TRIAC tanpa perhitungan LHS, dengan LHS dan tanpa argumen yang berarti telah ada *file input* yang terdefinisi lengkap dengan tambahan untuk *meshing* interpolasi.

Proses selanjutnya adalah membaca informasi dari *file input*. Ada lima informasi yang harus diperoleh dari *file input*, masing-masing adalah informasi geometri, karakteristik material, sejarah irradiasi dan kecelakaan serta rentang pengukuran temperatur saat irradiasi. Setelah data-data tersebut diperoleh, langkah selanjutnya adalah perhitungan geometri partikel triso. Kemudian, agar perhitungan dapat dilakukan, perlu dilakukan inisiasi obyek dari *class* `core.py`. Setelah obyek tersebut diinisiasi, semua perhitungan seperti ditunjukkan dalam Gambar 2.1 dilakukan.

Seperti telah dijelaskan dalam sub bab 3.1, TRIAC menjalankan perhitungan pada kondisi irradiasi (Gambar 3.1) dan kecelakaan (Gambar 3.2). Perhitungan pada kondisi irradiasi akan menghasilkan T_B yang secara bertahap diperoleh dari *mesh* sejarah irradiasi kemudian OPF. Selanjutnya, nilai T_B akan digunakan untuk menghitung τ_i, σ_0 dan m_0. Kemudian, perhitungan di kondisi irradiasi juga menghasilkan volume setiap layer dalam partikel triso, khususnya volume dua lapisan terdalam (kernel dan buffer, lihat Gambar 1.1). Nilai-nilai tersebut digunakan dalam perhitungan ketika kondisi kecelakaan terjadi, hingga akhirnya diketahui fraksi gagal partikel triso.

Yang menarik adalah model yang dikembangkan oleh PANAMA di kondisi kecelakaan yang melakukan perhitungan semua parameter di setiap *mesh* sejarah kecelakaan. Hal ini berbeda ketika perhitungan dilakukan di kondisi irradiasi, hanya diperoleh nilai antara untuk



Gambar 3.5: Interaksi antar fungsi untuk mendapatkan nilai tekanan yang dialami lapisan silikon karbida

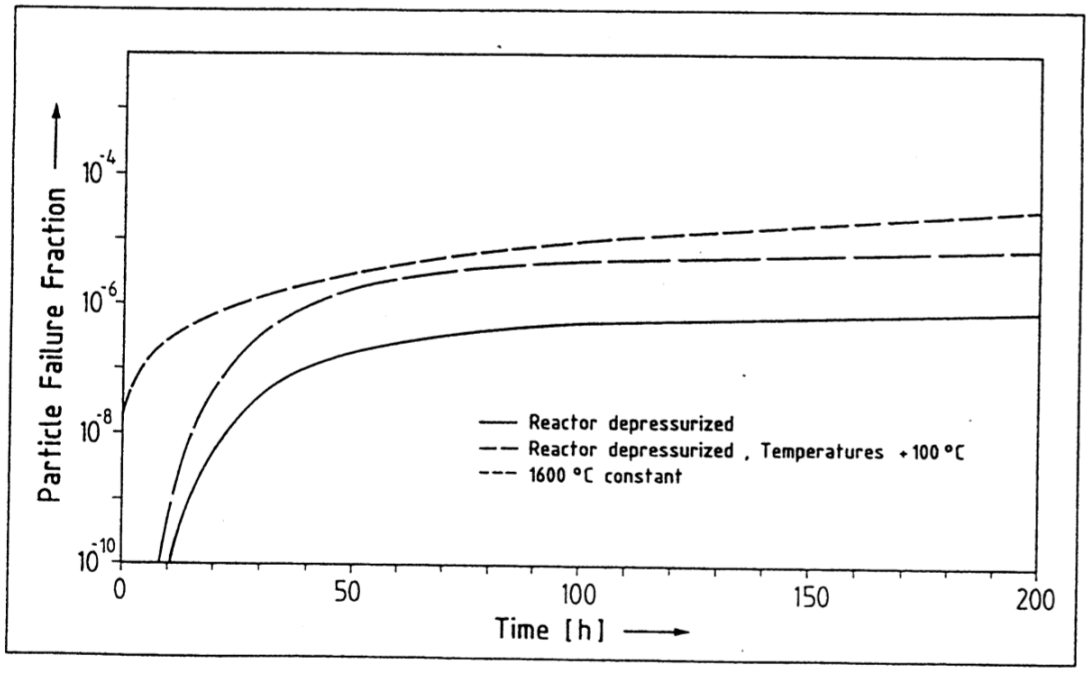
selanjutnya diperoleh hasil akhir berupa T_B. Dari T_B inilah nantinya diperoleh τ_i, σ_0 dan m_0. Kondisi ini sejalan dengan model pertumbuhan fraksi gagal ϕ_1 yang selalu mengacu pada kondisi awal kecelakaan (akhir irradiasi). Seperti ditunjukkan di Gambar 3.5, selalu ada parameter T_B, τ_i dalam perhitungan nilai OPF kecelakaan, Fd dan tekanan internal (p). Juga σ_0 dan m_0 dalam perhitungan σ_t dan ϕ_1 (Gambar 3.2).

BAB 4

Pengujian perhitungan TRIAC

4.1 Pendahuluan

Pengujian dilakukan dengan membandingkan hasil perhitungan TRAIC dengan PANAMA untuk data *file input* seperti dalam Lampiran 6.2. Khusus untuk hasil PANAMA, hanya terdapat hasil plot kondisi pengujian yang terdapat dalam dokumen teknis [5] (Gambar 4.1). Dari gambar tersebut, direkonstruksi titik-titik hubungan antara waktu (jam) di sumbu (x) terhadap ϕ_1 di sumbu (y). Karena itu, hasil pengujian TRIAC akan sangat tergantung dengan ketelitian dalam merekonstruksi titik tersebut. Pengujian yang dilakukan telah disajikan dalam artikel [11].

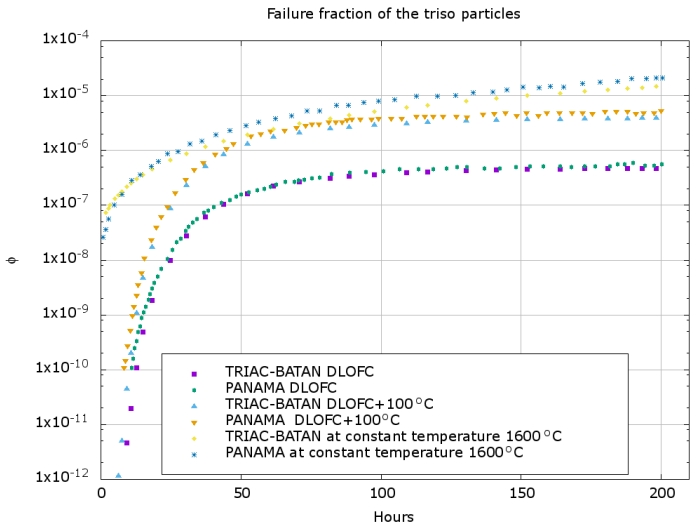


Gambar 4.1: Hasil perhitungan PANAMA untuk berbagai kondisi pengujian [5]

4.2 Hasil pengujian

Seperti ditunjukkan Gambar 4.1, kondisi pengujian adalah sebagai berikut. Hasilnya ditunjukkan pada Gambar 4.2.

1. Kondisi kecelakaan DLOFC (*Depressurized Loss Of Forced Cooling*) seperti kondisi *mesh* sejarah kecelakaan yang sama dengan data di Lampiran 6.2
2. Kondisi DLOFC dengan *mesh* sejarah kecelakaan sebelumnya ditambah 100^oC
3. Kondisi *mesh* sejarah kecelakaan yang stabil di angka 1600^oC



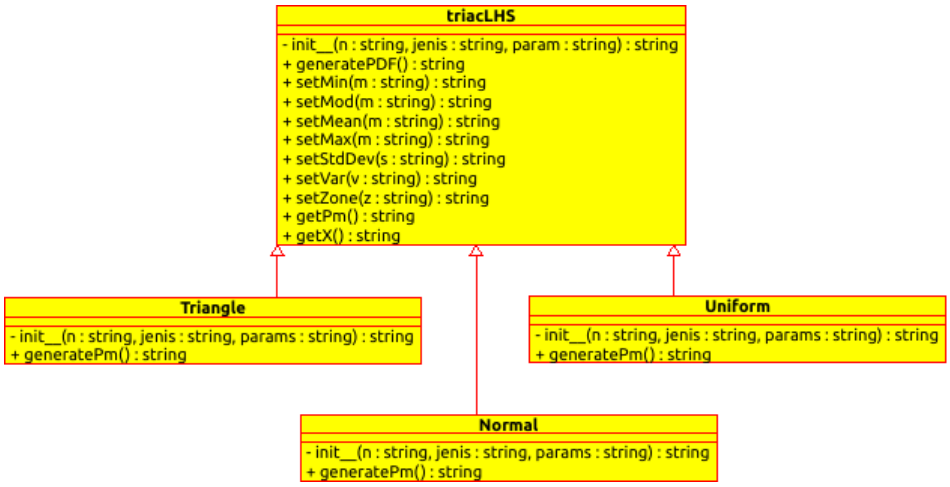
Gambar 4.2: PANAMA vs. TRIAC-BATAN for three accident scenarios

BAB 5

Penerapan modul *sampling*

5.1 Pendahuluan

Modul sampling yang akan diintegrasikan dalam TRIAC memiliki struktur class seperti pada Gambar 5.1. Setiap distribusi menerapkan perhitungannya masing-masing untuk mendapatkan variabel sampling. Sedangkan class `triacLHS` sebagai super class menangani semua layanan lain yang berlaku untuk semua distribusi.

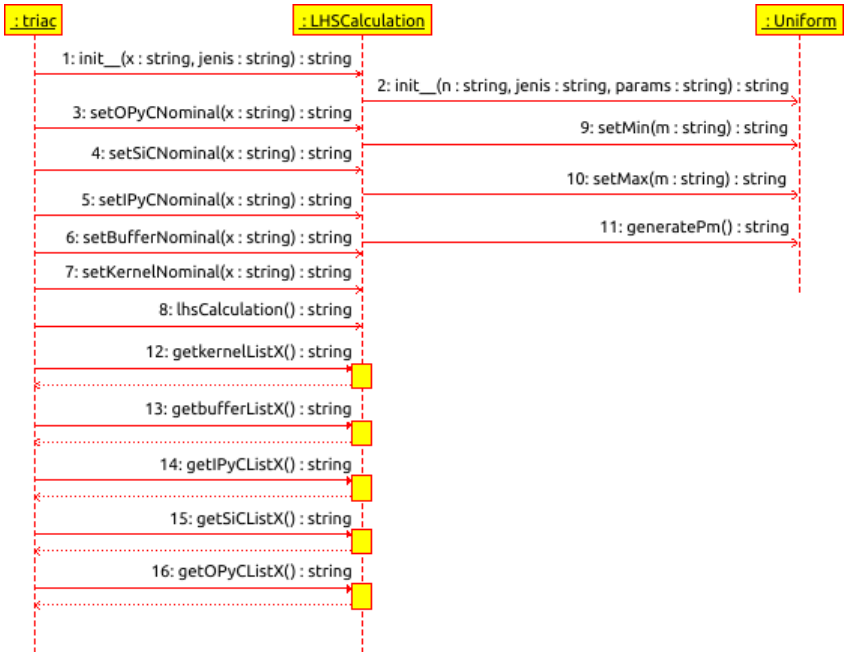


Gambar 5.1: Keterkaitan antar class dalam modul *sampling*

Dengan struktur TRIAC yang telah dibangun sebelumnya (Bab 3), diperlukan alur eksekusi yang dapat mengintegrasikan keduanya seperti yang disajikan di Gambar 2.3. Selanjutnya, agar dapat diterapkan diperlukan skenario eksekusi layanan antar class seperti yang ditunjukkan pada Gambar 5.2. Dalam Gambar 5.2, class `LHSCalculation` berperan sebagai antarmuka untuk memanfaatkan layanan yang disediakan modul *sampling*. Melalui class `LHSCalculation`, argumen yang diberikan oleh pengguna seperti jumlah *sample* yang dibutuhkan, jenis distribusi dan *sampling*, layer triso mana saja yang perlu di-*sampling* disampaikan ke modul *sampling*. Class `LHSCalculation` juga menjadi perantara untuk *sample* dihasilkan dan selanjutnya dikembalikan ke TRIAC.

Jenis data `string` yang terlihat dalam rangkaian eksekusi layanan di Gambar 5.2 adalah hasil transformasi kode sumber python oleh aplikasi UML modeler, Umbrello [1]. Meskipun

[1] `https://umbrello.kde.org/`



Gambar 5.2: Diagram *sequence* eksekusi layanan antar class yang melibatkan modul *sampling*

umumnya berupa list yang berisi informasi ketidakpastian pada lapisan triso. Informasi ketidakpastian yang akan digunakan dalam pengujian TRIAC dengan modul *sampling* diberikan pada Tabel 5.1.

Informasi terkait ketidakpastian tersebut dimasukkan ke TRIAC melalui *file input* seperti pada Lampiran 6.2, di bawah baris informasi ”Uncertainties in geometry and a number of samples:”. Untuk setiap layer, terdapat 3 nilai yang harus diberikan, masing-masing adalah nilai ketidakpastian, jenis distribusi serta standar deviasi (σ) jika jenis distribusi yang dipilih adalah normal. Jika jenis distribusi yang dipilih bukan normal, maka nilai σ diisi dengan 0. Demikian juga jika lapisan triso tertentu, atau bahkan semuanya, tidak dipertimbangkan ketidakpastiannya, di bagian itu diisi nilai 0. Sementara nilai terakhir di barisan itu adalah jumlah *sample*. Modul sampling ini menerapkan kebijakan untuk menggunakan jumlah *sample* yang sama untuk semua layer triso.

Kembali ke Gambar 2.3, penggunaan modul *sampling* akan membuat perhitungan TRIAC melakukan perulangan sebanyak jumlah *sample*. Tentunya setelah sample telah dihasilkan oleh modul *sampling*. Karena ada sebanyak jumlah *sample* kombinasi ketebalan lapisan triso, maka akan ada sebanyak itu pula hasil perhitungan fraksi gagal triso. Itu sebabnya, di akhir perhitungan yang dijelaskan dalam diagram aktifitas di Gambar 2.3 ada perhitungan rerata dan variance dari nilai fraksi gagal yang ada sebanyak jumlah *sample*.

Tabel 5.1: Nilai nominal dan ketidakpastiannya pada lapisan triso

OPyC		SiC		IPyC		Buffer		Kernel	
Nominal	Ketidakpastian	Nominal	Ketidakpastian	Nominal	Ketidakpastian	Nominal	Ketidakpastian	Nominal	Ketidakpastian
4.60^{-4}	1.0^{-5}	4.20^{-4}	2.6^{-6}	3.85^{-4}	1.0^{-5}	3.45^{-4}	4.4^{-6}	2.55^{-4}	5.0^{-6}

5.2 Pembacaan *file input*

Pembacaan informasi terkait ketidakpastian ketebalan lapisan triso dilakukan di kode yang sama seperti informasi lainnya (Listing 1). Seperti telah dijelaskan di Sub bab 5.1, setiap lapisan partikel triso akan memiliki 3 data, masing-masing adalah ketidakpastian, jenis distribusi *sample* dan σ. Ketiga data tersebut kemudian disusun dalam 1 list. List berisi data ketidakpastian setiap layer selanjutnya digabungkan dengan data sejenis dari layer yang lain. Sedangkan jumlah *sample* yang terlibat menjadi data yang terakhir yang disimpan dalam list utama yang diberi nama `uncertainties` (baris ke-63 di Listing 1).

5.3 Trigger ke modul *sampling*

Seperti telah dijelaskan pada Gambar 5.2, permintaan layanan pada modul *sampling* dilakukan dari class `triac.py` (baris ke-224 dari Listing 4). Terdapat 2 argumen yang harus disertakan saat melakukan inisiasi obyek `LHSCalculation`. Keduanya adalah data ketidakpastian yang diperoleh dari pembacaan *file input* (sub bab 5.2) serta informasi tentang apakah akan menggunakan metode *sampling* LHS atau SRS.

Argumen yang dilewatkan ke `LHSCalculation` akan dipecah dan dimasukkan ke variabel yang bersesuaian. Sebagai contoh, elemen pertama dari list `uncertainties` adalah data ketidakpastian dari lapisan OPyC. Demikian untuk selanjutnya. Kondisi ini dapat dilihat pada fungsi `__init__` pada `LHSCalculation` di Listing 5.

Setelah inisiasi obyek `LHSCalculation`, `triac.py` selanjutnya memasukkan nilai nominal dari ketebalan seluruh lapisan triso. Proses ini dijelaskan dalam Gambar 5.2 sebagai pemanggilan fungsi ke-3,4,5,6 dan 7, masing-masing untuk lapisan OPyC, SiC, IPyC, buffer dan kernel.

Kemudian, jika distribusi yang dipilih adalah *uniform* maka `LHSCalculation` harus memasukkan nilai minimum dan maksimum dari rentang ketidakpastian. `LHSCalculation` mengetahui apa yang harus dilakukan karena sebelumnya, `triac.py` menjalankan perintah `lhsCalculation` (baris ke-235 pada Listing 4). Perintah tersebut akan menjalankan layanan yang tersedia di `LHSCalculation`, tepatnya layanan yang didefinisikan pada baris ke-93 pada Listing 5. Di dalam layanan tersebutlah, ditentukan inisiasi obyek lhs, apakah dari clas `Uniform`, `Triangle` ataupun `Normal`. Setelah penentuan obyek tersebut, pemberian nilai awal dilakukan, yang dalam Gambar 5.2 dilakukan pada pemanggilan fungsi ke-9 dan 10. Setiap obyek distribusi akan memiliki nilai awal yang berbeda. Hal ini menyebabkan definisi dari layanan `lhsCalculation` cukup panjang untuk mengakomodasi semua opsi yang mungkin.

Layanan terakhir yang pada Gambar 5.2 dijalankan oleh LHSCalculation adalah `generatePm`. Meskipun bernama sama, penerapannya berbeda-beda dari satu distribusi ke distribusi lainnya. Karena itu, penerapannya dilakukan di class distribusi. Hasil dari pemanggilan fungsi ini adalah *sample* dengan kriteria jumlah dan distribusi seperti yang diinginkan. Setelah layanan tersebut selesai, obyek `triac` dapat menggunakan *sample* tersebut untuk menghitung fraksi gagal triso. Selesainya layanan tersebut ditandai dengan pengambilan *sample* oleh obyek `triac` dari `LHSCalculation`. Pengambilan sample tersebut dalam Gambar 5.2 direpresentasikan oleh pemanggilan fungsi ke-12 s/d 16.

BAB 6

Pengujian modul *sampling*

6.1 Pendahuluan

Pengujian modul sampling akan mengunakan data sejarah irradiasi dan kecelakaan yang sama dengan yang telah digunakan pada pengujian perhitungan utama TRIAC. Sedangkan data ketidakpastian adalah seperti Tabel 5.1. Pengujian diawali dengan membuat sample dengan mengikuti distribusi tertentu. Dalam hal ini adalah distribusi *uniform* dan triangular. Sedangkan modul untuk distribusi normal yang telah diimplementasikan masih menghasilkan pola yang berbeda dari yang seharusnya. Jika telah dapat terbentuk, maka penguna annya akan sama seperti modul dengan distribusi *uniform* dan triangular.

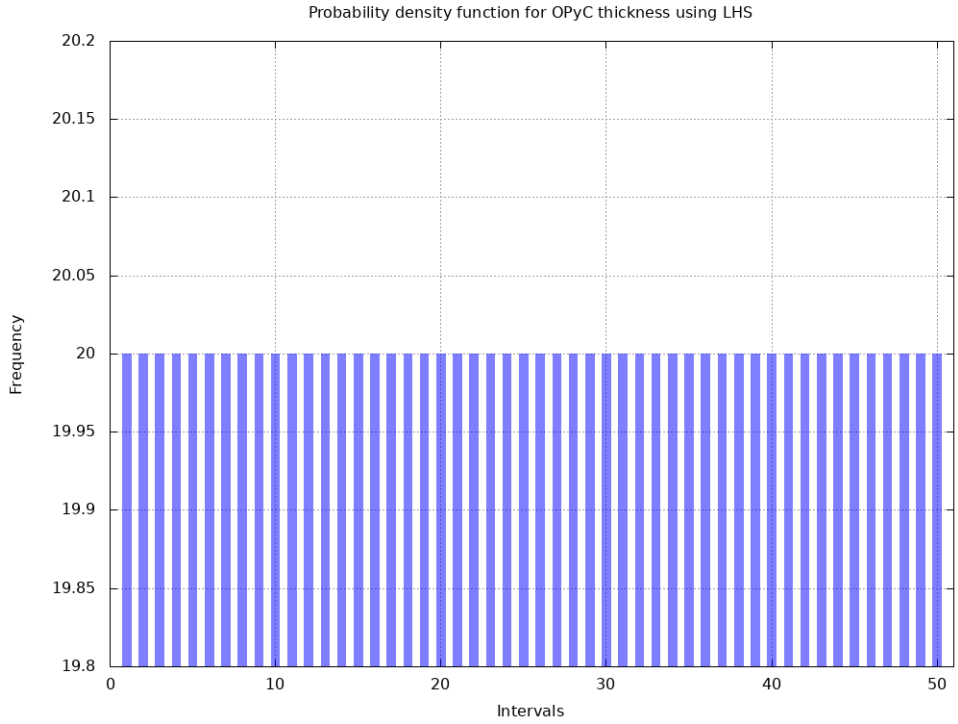
6.2 Hasil pengujian

Pengujian pertama adalah membentuk sample sehingga memiliki distribusi tertentu, baik menggunakan metode LHS dan SRS. Gambar 6.1(a) dan 6.1(b) menunjukkan *sample* yang terbentuk mengikuti distribusi *uniform*, dengan menggunakan metode LHS dan SRS. Sedangkan Gambar 6.2(a) dan 6.2(b) menunjukkan pola serupa dengan distribusi triangular. Terlihat bahwa metode LHS dapat membentuk sample yang jumlahnya terdistribusi dengan distribusi tertentu. Sedangkan SRS, cukup acak dalam distribusinya.

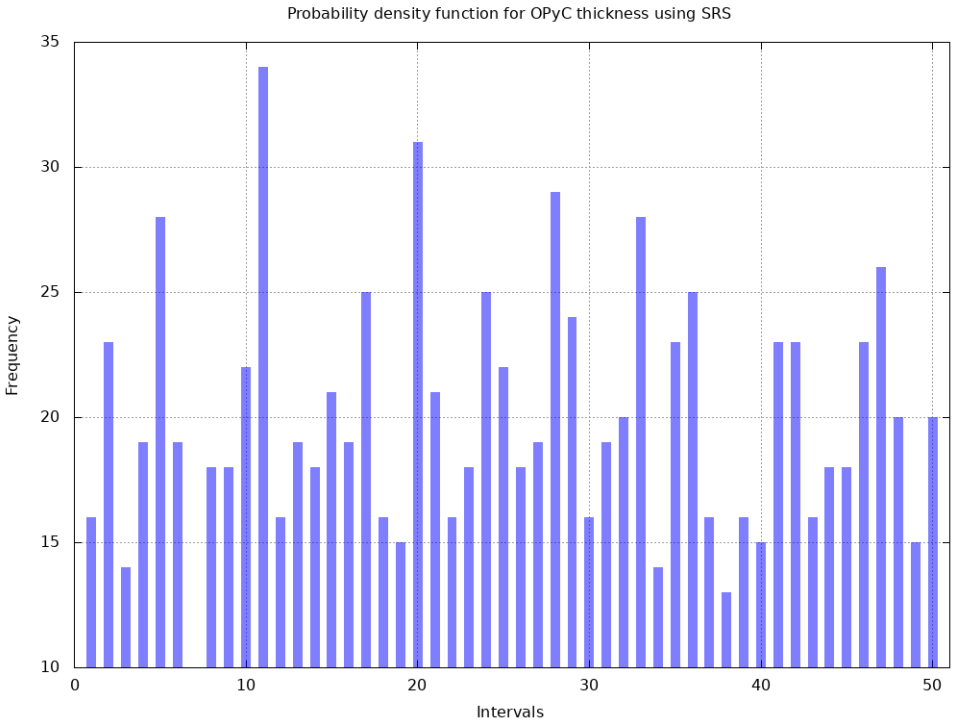
Pengujian selanjutnya adalah untuk melihat dampak dari mempertimbangkan ketidakpastian pada setiap lapisan triso yang seragam dalam hal distribusi dan metode *sampling*. Yang dimaksud dengan seragam tersebut adalah, jika skenario yang sedang dijalankan adalah menggunakan distribusi *uniform* dengan metode sampling LHS, maka semua lapisan triso akan menggunakan distribusi dan metode *sampling* yang sama. Kemudian, hasilnya akan disajikan dalam bentuk plot nilai rerata fraksi gagal dan penyimpangan bakunya (σ). Skenario ini seluruhnya dijalankan dengan menggunakan 100 *sample*.

Gambar 6.3(a) dan 6.3(b) menunjukkan pola fraksi gagal yang diperoleh dari *sample* yang diperoleh menggunakan distribusi *uniform*, baik dengan metode LHS maupun SRS. Sedangkan Gambar 6.4(a) dan 6.4(b) adalah hasil dari pengujian serupa menggunakan *sample* yang terdistribusi triangular.

Dari perspektif distribusi *sample*, diketahui bahwa distribusi *uniform* menghasilkan σ yang lebih besar daripada triangular. Hal ini dapat dipahami ketika kita melihat Gambar 6.2(a) dan 6.2(b). Pada gambar dengan distribusi triangular tersebut, *sample* yang berukuran tidak jauh dari nilai rerata adalah *sample* dengan jumlah yang mayoritas. Secara statistik hal ini disebut sebagai kondisi di mana variasi *sample* rendah. Hal ini berkebalikan dengan distribusi *uniform* yang *sample* yang terbentuk terdistribusi cukup seragam dalam rentang ketidakpastian.

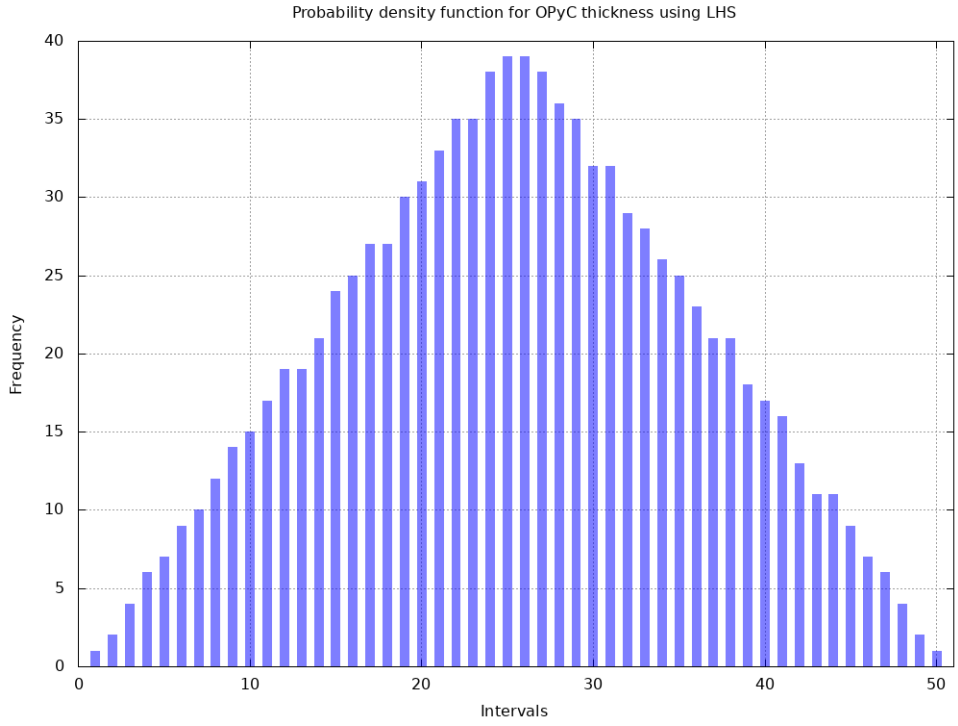


(a)

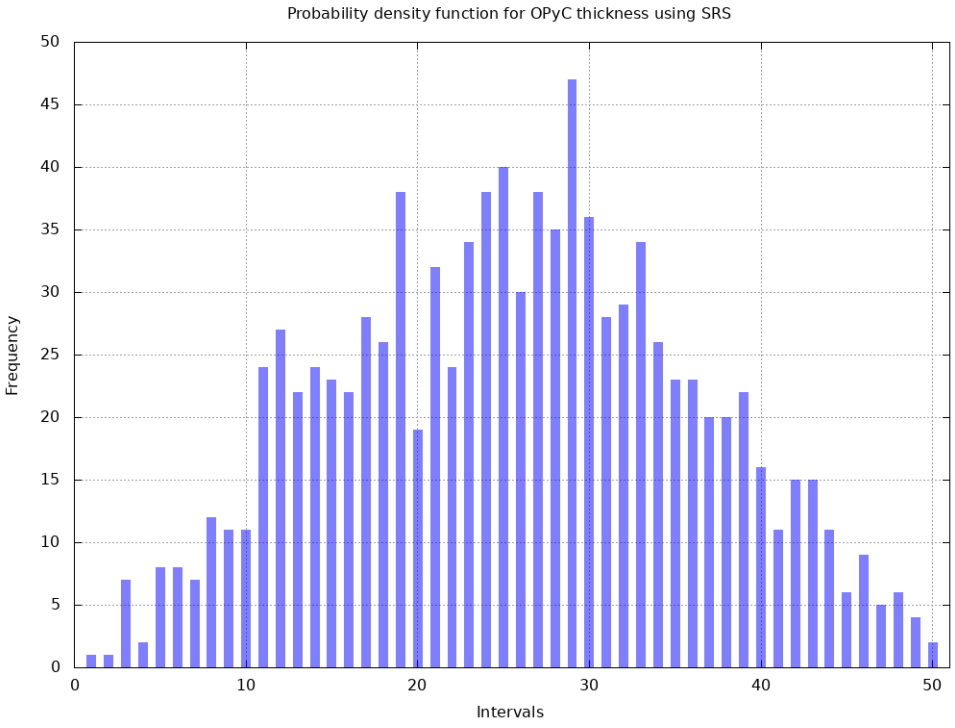


(b)

Gambar 6.1: Distribusi *sample* yang diperoleh menggunakan distribusi *uniform* serta metode (a). LHS dan (b). SRS

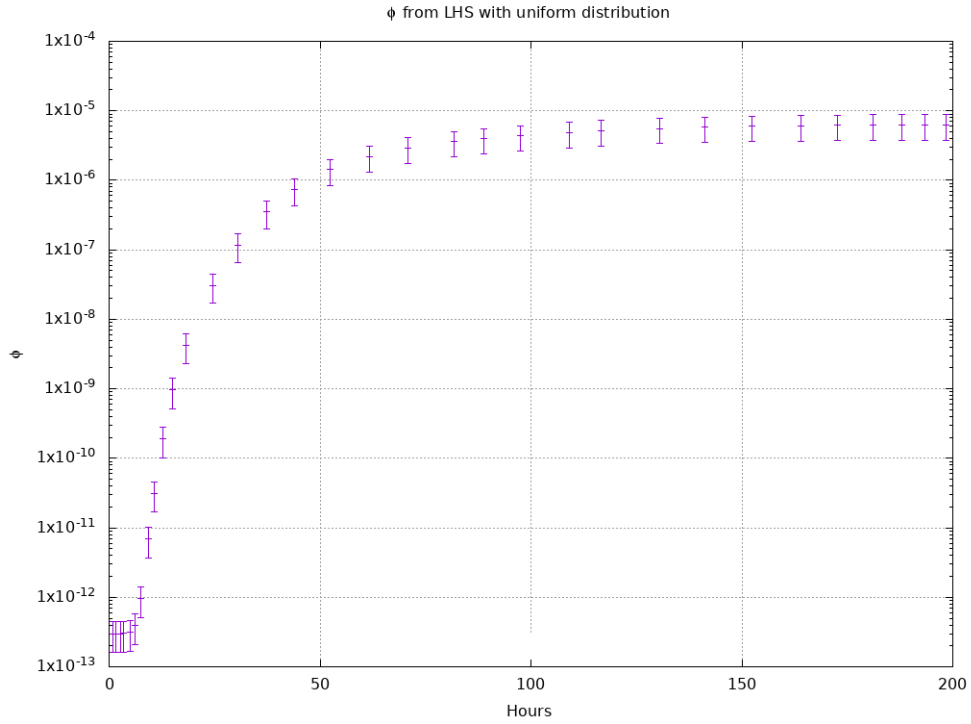


(a)

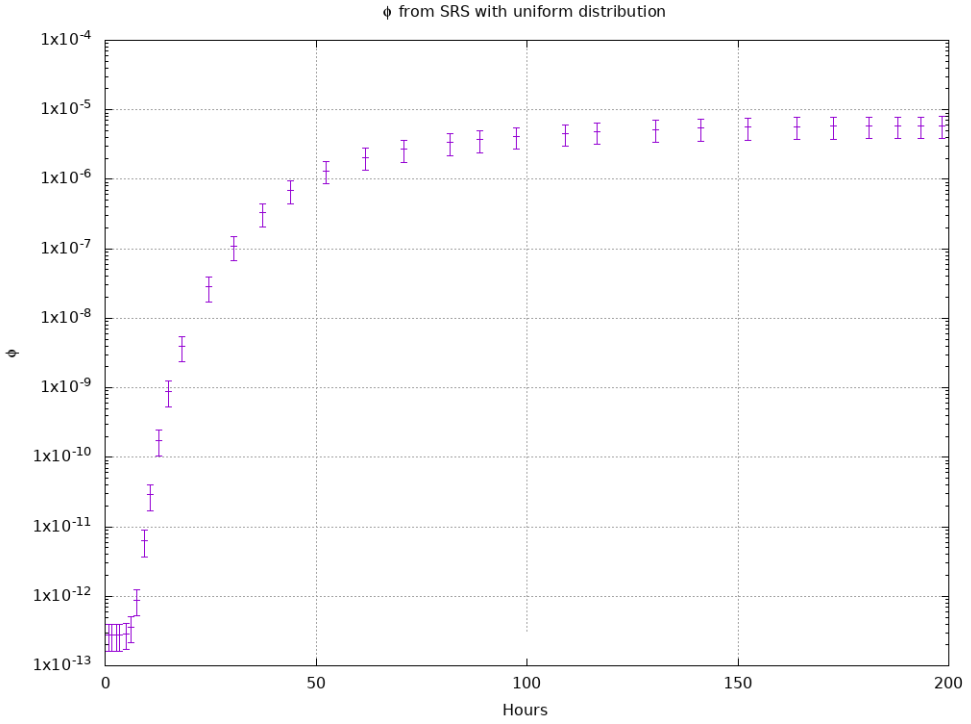


(b)

Gambar 6.2: Distribusi *sample* yang diperoleh menggunakan distribusi triangular serta metode (a). LHS dan (b). SRS

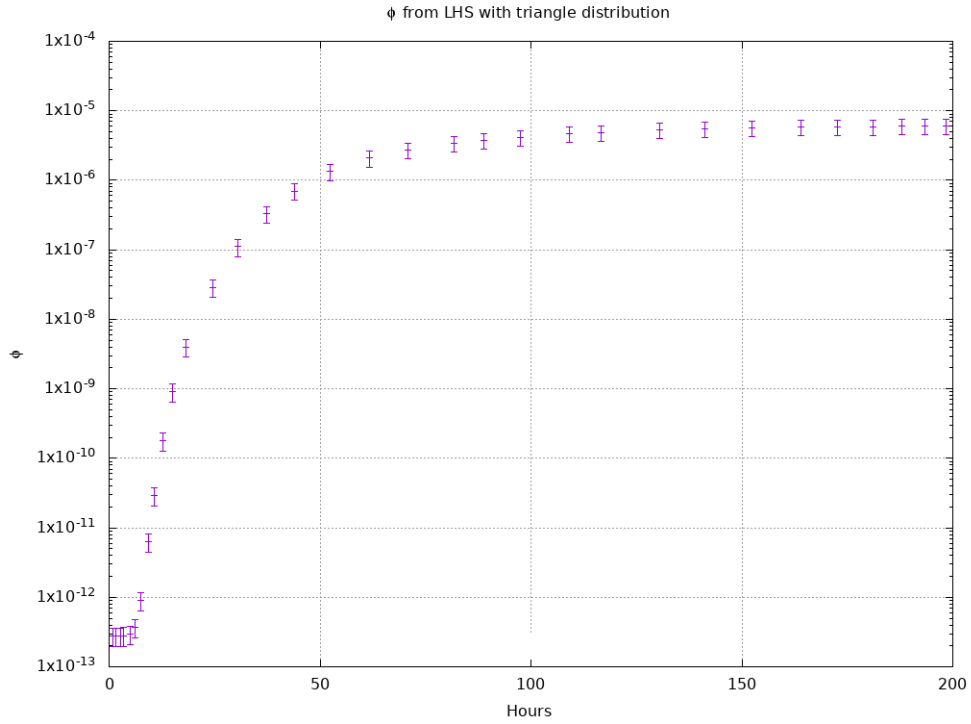


(a)

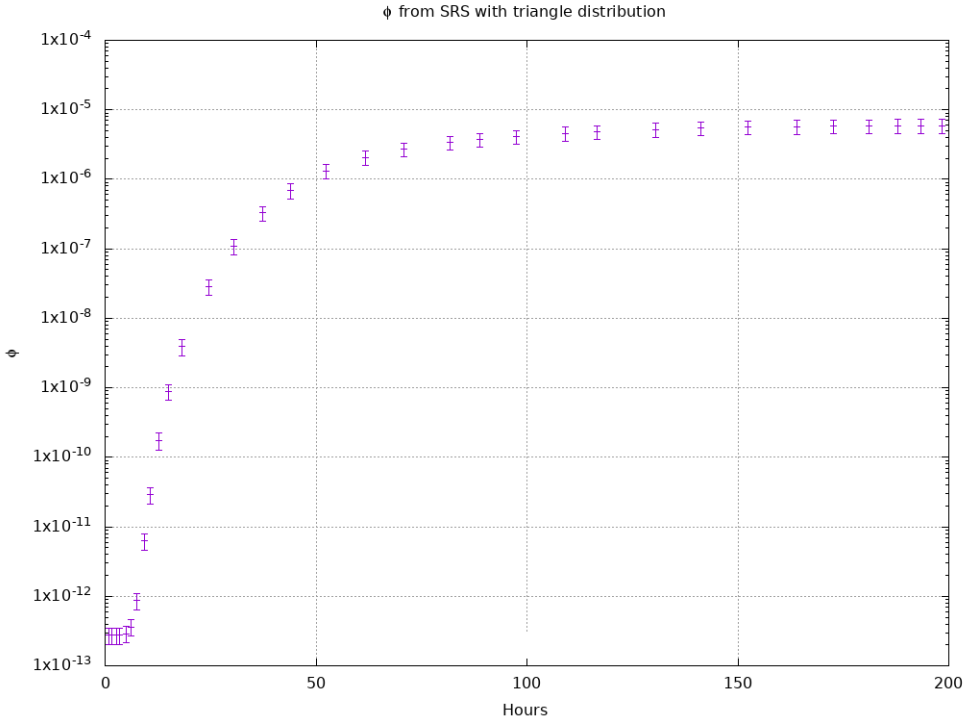


(b)

Gambar 6.3: Nilai rerata dan σ fraksi gagal partikel triso menggunakan metode σ (a). LHS and (b). SRS dan distribusi *uniform*



(a)



(b)

Gambar 6.4: Nilai rerata dan σ fraksi gagal partikel triso menggunakan metode σ (a). LHS and (b). SRS dan distribusi triangular

Sedangkan dari perspektif metode sampling, LHS justru menghasilkan nilai σ yang cenderung lebih besar daripada SRS. Hal ini dapat dipahami dari Gambar 6.1(a) dan 6.1(b). Pada distribusi *sample* yang diperoleh menggunakan SRS terdistribusi secara acak, sehingga ada rentang ketidakpastian yang tidak memiliki *sample*. Kondisi seperti ini memungkinkan terjadinya efek saling menghilangkan variasi sehingga menghasilkan nilai σ cenderung kecil.

Hasil terkahir memang belum valid karena fraksi gagal diperoleh dengan mensimulasi ketidakpastian secara bersamaan di setiap lapisan triso. Masih perlu dievaluasi pengaruh ketidakpastian di setiap lapisan secara terpisah terhadap fraksi gagal triso.

Daftar Referensi

[1] J. Wang, "An integrated performance model for high temperature gas cooled reactor coated particle fuel," Ph.D. dissertation, Massachusetts Institute of Technology, 2004.

[2] "Reaktor daya eksperimental (rde)," http://www.batan.go.id/index.php/id/reaktor-daya-eksperimental-rde, diakses: 17-07-2017.

[3] T. Setiadipura, D. Irwanto, and Zuhair, "Preliminary neutronic design of high burnup otto cycle pebble bed reactor," *Atom Indonesia*, vol. 41, no. 1, pp. 7–15, 2015.

[4] K. Verfondern, J. Cao, T. Liu, and H.-J. Allelein, "Conclusions from v&v studies on the german codes panama and fresco for htgr fuel performance and fission product release," *Nuclear Engineering and Design*, vol. 271, pp. 84 – 91, 2014, sI : HTR 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0029549313005992

[5] K. Verfondern and H. Nabielek, "The mathematical basis of the panama-i code for modelling pressure vessel failure of triso coated particles under accident conditions," Julich Research Center, Germany, Tech. Rep., 1990.

[6] J. Helton and F. Davis, "Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems," *Reliability Engineering & System Safety*, vol. 81, no. 1, pp. 23 – 69, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832003000589

[7] M. D. Shields and J. Zhang, "The generalization of latin hypercube sampling," *Reliability Engineering & System Safety*, vol. 148, pp. 96 – 108, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0951832015003543

[8] "Latin hypercube sampling: Simple definition," http://www.statisticshowto.com/latin-hypercube-sampling/, diakses: 19-03-2018.

[9] W. K. Terry, L. M. Montierth, S. S. Kim, J. J. Cogliati, and A. M. Ougouag, "Evaluation of the initial critical configuration of the htr-10 pebble-bed reactor," Idaho National Lab, US, Tech. Rep., 2006.

[10] N. A. Wahanani, "Dokumentasi program latin hypercube sampling untuk analisa ketidakpastian," bidang Komputasi, Pusat Pengembangan Informatika Nuklir - BATAN.

[11] A. A. Waskita and T. Setiadipura, "The development of triac-batan: a triso fuel performance analysis code," in *Symposium of Emerging Nuclear Technology and Engineering Novelty*, 2018.

LAMPIRAN

Lampiran 1: Contoh file input

TRIAC-BATAN
TRISO Analysis Code of BATAN
"Developed by Computational Laboratory, Center for Nuclear Reactor Technology and Safety, BATAN"

Case Title: (describe your problem case here)

TRISO Geometry:

Outer radius	CFP	SiC	IPyC	buffer	kernel	center
[m]	4.60E-04	4.20E-04	3.85E-04	3.45E-04	2.55E-04	0

Uncertainties in geometry and a number of samples:

1.0E-5	1	0.0	2.6E-6	1	0.0	1.0E-5	1	0.0	4.4E-6
1	0.0	5.0E-6	1	0.0	100				

Properties and Operation Parameters:

SiC Tensile Strength [Pa]	Weibull Modulus	Burnup [FIMA]	"Fission Yield of stable fission gasses, Ff"	Fast Neutron Fluence	Weight ratio of th to U-235 in kernel
8.34E+08	8.02	0.08	0.31	1.4	

Properties and Operation Parameters related with thermal decomposition:

INPUT: Irradiation Temp. Hystory

1	0	593
2	17	833
3	34	1023
4	51	1093
5	68	1123
6	85	593
7	102	833
8	119	1023
9	136	1093
10	153	1123
11	170	593
12	187	833
13	204	1023
14	221	1093
15	238	1123
16	255	593
17	272	833
18	289	1023
19	306	1093
20	323	1123
21	340	593
22	357	833
23	374	1023
24	391	1093
25	408	1123
26	425	593
27	442	833
28	459	1023
29	476	1093
30	493	1123
31	510	593
32	527	833
33	544	1023
34	561	1093
35	578	1123
36	595	593
37	612	833
38	629	1023
39	646	1093
40	663	1123
41	680	593
42	697	833
43	714	1023
44	731	1093
45	748	1123
46	765	593
47	782	833
48	799	1023
49	816	1093
50	833	1123
51	850	593

52	867	833
53	884	1023
54	901	1093
55	918	1123
56	935	593
57	952	833
58	969	1023
59	986	1093
60	1003	1123
61	1020	593

INPUT: Accident Temp. Hystory

1	0.959232614	973
2	1.678657074	934.5384615
3	1.698657074	996.0769231
4	2.637889688	1053.769231
5	3.357314149	1157.615385
6	5.035971223	1253.769231
7	6.23501199	1334.538462
8	7.434052758	1423
9	9.352517986	1492.230769
10	10.79136691	1561.461538
11	12.70983213	1615.307692
12	15.10791367	1661.461538
13	18.22541966	1696.076923
14	24.70023981	1742.230769
15	30.45563549	1765.307692
16	37.41007194	1773
17	43.88489209	1780.692308
18	52.27817746	1769.153846
19	61.63069544	1757.615385
20	70.74340528	1742.230769
21	81.77458034	1723
22	88.72901679	1715.307692
23	97.60191847	1703.769231
24	109.1127098	1692.230769
25	116.5467626	1684.538462
26	130.4556355	1665.307692
27	141.2470024	1649.923077
28	152.2781775	1638.384615
29	164.028777	1623
30	172.6618705	1611.461538
31	181.0551559	1599.923077
32	188.0095923	1592.230769
33	193.2853717	1588.384615
34	198.3213429	1573

Lampiran 2: InputData.py

Listing 1: InputData.py

```
1  import sys, math, re
2  def readdata(namafile):
3          f=open(namafile,"r")
4          statusGeometry="[m]"
5          statusUncertainties="Uncertainties in geometry"
6          statusCharacteristics="SiC Tensile Strength [Pa]"
7          statusIrradiation="INPUT: Irradiation Temp. Hystory"
8          statusAccident="INPUT: Accident Temp. Hystory"
9          statusAll=0
10         dimensi=[]
11         uncertainties=[]
12         characteristics=[]
13         irradiation=[]
14         accident=[]
15         i=0
16         x=0
17         a=0.0
18         b=0.0
19         c=0
20         for baris in f.readlines():
21                 i=i+1
22                 element=baris.split('\t')
23                 if len(element)!=0:
24                         if statusAll==0:
25                                 if statusGeometry in baris:
```

```
                for j in range(1,7):
                        y=float(element[j])
                        dimensi.append(y)
                statusAll=1

elif statusAll==1:
        if statusUncertainties in baris:
                x=i+1
        elif i==x:
                temp=[]
                temp.append(float(element[0]))
                temp.append(int(element[1]))
                temp.append(float(element[2]))
                uncertainties.append(temp)
                temp=[]
                temp.append(float(element[3]))
                temp.append(int(element[4]))
                temp.append(float(element[5]))
                uncertainties.append(temp)
                temp=[]
                temp.append(float(element[6]))
                temp.append(int(element[7]))
                temp.append(float(element[8]))
                uncertainties.append(temp)
                temp=[]
                temp.append(float(element[9]))
                temp.append(int(element[10]))
                temp.append(float(element[11]))
                uncertainties.append(temp)
                temp=[]
                temp.append(float(element[12]))
                temp.append(int(element[13]))
                temp.append(float(element[14]))
                uncertainties.append(temp)
                temp=[]
                d=element[15].split('\n')
                temp.append(int(d[0]))
                uncertainties.append(temp)
                for i in range(6):
                        print(uncertainties[i])
                statusAll=2
```

```
67
68                  elif statusAll==2:
69                      if statusCharacteristics in baris:
70                              x=i+1
71                      elif i==x:
72                              try:
73                                      for j in range(0,5):
74                                              y=float(element[j])
75                                              characteristics.append(y)
76                                      statusAll=3
77                              except:
78                                      x=i+1
79
80                  elif statusAll==3:
81                      if statusIrradiation in baris:
82                              x=i+1
83                      elif i==x:
84                              if statusAccident in baris:
85                                      statusAll=4
86                              else:
87                                      temp=[]
88                                      try:
89                                              l=float(element[1])
90                                              c=c+1
91                                              if l>=0:
92                                                      temp.append(l*24*3600)
93                                                      if c==1:
94                                                              a=1
95                                                      if c==2:
96                                                              b=1
97                                                      m=float(element[2])
98                                                      temp.append(m)
99                                                      irradiation.append(temp)
100                                             x=i+1
101                                     except:
102                                             x=i+1
103
104                 elif statusAll==4:
105                     temp=[]
106                     try:
107                             y=float(element[1])*3600
```

```
108                                         temp.append(y)
109                                         y=float(element[2])
110                                         temp.append(y)
111                                         accident.append(temp)
112                                 except:
113                                         x=i+1
114
115         f.close()
116         return dimensi, uncertainties, characteristics, irradiation, accident, b-a
```

Lampiran 3: interpolasi.py

Listing 2: Interpolasi.py

```
def linier(a,b,c):
        x1=a[0]
        y1=a[1]
        x2=b[0]
        y2=b[1]

        i=[]
        selisih=(x2-x1)/c
        x=x1

        for k in range(1,c):
                j=[]
                x=x+selisih
                y=(((x-x1)/(x2-x1))*(y2-y1))+y1
                j.append(x)
                j.append(y)
                i.append(j)
        return i
```

Lampiran 4: core.py

Listing 3: core.py

```
1  import math
2  class CORE:
3          def __init__(self):
4                  self.R=8.3143
5                  self.Ff=0.0
6                  self.Fb=0.0
7                  self.Vm=0.0
8                  self.d0=0.0
9                  self.m0=0.0
10                 self.m00=0.0
11                 self.r=0.0
12                 self.sigma0=0.0
13                 self.sigma00=0.0
14                 self.Vk=0.0
15                 self.Vf=0.0
16                 self.tb=0.0
17                 self.Tb=0.0
18                 self.Fluence=0.0
19
20         def OPF(self,irradiation,y):
21                 x=len(irradiation)
22                 print("Irradiation length:",x)
23                 z=0.0
24                 for i in range(x):
25                         j=irradiation[i]
```

```
                    a1=self.R*j[1]
                    a=-163000/(a1)
                    b=math.exp(a)
                    g=2*(8.32e-11)*b
                    g1=g*(self.tb-j[0])*y
                    z=z+g1
            return z

    def setFf(self, ff):
            self.Ff=ff

    def setM00(self,M00):
            self.m00=M00

    def setFb(self,fb):
            self.Fb=fb

    def setVm(self,vm):
            self.Vm=vm

    def setRD0(self,a,b):
            x=(0.5*(math.pow(a,3)+math.pow(b,3)))
            self.r=math.pow(x,(1.0/3))
            self.d0=a-b

    def setSigma00(self,s0):
            self.sigma00=s0

    def setSigma0(self):
            logGammaS=0.556+(650/self.Tb)
            gammaS=math.pow(10,logGammaS)
            self.sigma0=self.sigma00*(1-(self.Fluence/gammaS))

    def setM0(self):
            logGammaM=0.394+(650/self.Tb)
            gammaM=math.pow(10,logGammaM)
            self.m0=self.m00*(1-(self.Fluence/gammaM))
            """self.m0=6.93"""

    def setVk(self,vk):
            self.Vk=vk
```

```
67
68          def setVf(self,vf):
69                  self.Vf=0.5*vf
70
71          def setFluence(self,f):
72                  self.Fluence=f
73
74          def set_tb(self,t_b):
75                  self.tb=t_b
76
77          def set_Tb(self, T_b):
78                  self.Tb=T_b
79
80          def getSigma0(self):
81                  return self.sigma0
82
83          def getM0(self):
84                  return self.m0
85
86          def getFluence(self):
87                  return self.Fluence
88
89          def FTau(self,tau):
90                  looping=0.0
91                  for n in range (1,2000):
92                          pangkat=math.pow(n,2)*math.pow(math.pi,2)*tau
93                          A=math.exp(-(pangkat))
94                          B=math.pow(n,4)*math.pow(math.pi,4)
95                          looping=looping+((1-A)/B)
96                          """looping=looping+(1-(A/B))"""
97                  ftau=1-((6/tau)*looping)
98                  return ftau
99
100         def OPFAccident(self,T):
101                 a=(2*math.log10(self.tb))
102                 b=(0.404*((10000/T)-(10000/(self.Tb+75))))
103                 logOPF=-10.08-(8500/self.Tb)+a-b
104                 opfa=math.pow(10,logOPF)
105                 return opfa
106
107         def DS(self,T):
```

```
            logDS=-2.3-(8116/T)
            ds=math.pow(10,logDS)
            return ds

    def volume(self,r):
            return (4/3)*math.pi*r*r*r

    def weibullParam(self,T,t):
            """a=-187400/(self.R*T)
            b=math.pow(math.e,a)
            etaDot=0.565*b
            c=math.pow(math.e,-etaDot*t)
            d=self.m0*(0.44+(0.56*c))
            print(t,T,d)"""
            return self.m0

    def tekanan(self,Fd,opf,T):
            p=((Fd*self.Ff)+opf)*self.Fb*(self.Vk/self.Vm)*self.R*T/self.Vf
            return p

    def SIGMA_T(self,p,t,T):
            nu=(5.87e-7)*math.exp(-179500/(self.R*T))
            a=(self.r*p/(2*self.d0))*(1+(nu*t/self.d0))
            """print(r,p,d,t,T,nu)"""
            return a,nu

    def PHI(self,sigmaT, m):
            a=sigmaT/self.sigma0
            b=math.pow(a,m)
            c=math.log(2)*b
            d=1-math.exp(-c)
            return d
```

Lampiran 5: triac.py

Listing 4: triac.py

```
1  import math,sys,shutil
2  from InputData import readdata
3  from Interpolasi import linier
4  import core as c
5  from LHScalculation import LHSCalculation
6
7
8  def triacCalculation(dimensi,characteristics,irradiation,accident):
9          utama=c.CORE()
10         lenIR=len(irradiation)
11         tb=irradiation[lenIR-1][0]
12         utama.set_tb(tb)
13         VolRef1=utama.volume(dimensi[0])
14         VolRef2=utama.volume(dimensi[1])
15         VolOPyC=VolRef1-VolRef2
16
17         """Volume SiC"""
18         VolRef3=utama.volume(dimensi[2])
19         VolSiC=VolRef2-VolRef3
20
21         """Volume IPyC"""
22         VolRef4=utama.volume(dimensi[3])
23         VolIPyC=VolRef3-VolRef4
24
25         """Volume Buffer & Volume Kernel"""
```

```
26          VolKernel=utama.volume(dimensi[4])
27          utama.setVk(VolKernel)
28          VolBuff=VolRef4-VolKernel
29          utama.setVf(VolBuff)
30          utama.setVm(2.43796e-5)
31
32          utama.setSigma00(characteristics[0])
33          utama.setM00(characteristics[1])
34          utama.setFb(characteristics[2])
35          utama.setFf(characteristics[3])
36          utama.setFluence(characteristics[4])
37
38          utama.setRD0(dimensi[1],dimensi[2])
39
40          dt=10
41          if dt>1:
42                  irradiation2=[]
43                  irradiation2.append(irradiation[0])
44                  b=1
45                  for x in range(1,lenIR):
46                          temp=[]
47
48                          i=irradiation[x-1][0]
49                          j=irradiation[x][0]
50                          if i!=j:
51                                  temp=linier(irradiation[x-1],irradiation[x],dt)
52                                  for y in range(len(temp)):
53                                          irradiation2.append(temp[y])
54                                          b=b+1
55                          irradiation2.append(irradiation[x])
56                          b=b+1
57
58                  irradiation=irradiation2
59                  irradiation2=[]
60                  y=(float(rentang)/dt)*24*3600
61                  opf=utama.OPF(irradiation,y)
62
63          Tb=0.85e4/((2*math.log10(tb))-(math.log10(opf))-10.08)
64          print('Tb='+str(Tb)+'\n')
65
66          utama.set_Tb(Tb)
```

```
67          utama.setSigma0()
68          utama.setM0()
69          print('sigma0='+str(utama.getSigma0())+', '+'m0='+str(utama.getM0()))
70
71          dsi=utama.DS(Tb)
72          tauI=dsi*tb
73          print('dsi='+str(dsi)+', '+'tauI='+str(tauI))
74          """Akhir dari proses irradiasi"""
75
76          phi1=0
77
78          """Tambahan dari triacc per tanggal 8-12-2017"""
79          dsa=utama.DS(accident[0][1])
80          tauA=dsa*accident[0][0]
81          if tauA==0:
82                  Fd=utama.FTau(tauI)
83          else:
84                  Fd=(((tauI+tauA)*utama.FTau(tauI+tauA))-(tauA*utama.FTau(tauA)))/tauI
85
86          p=utama.tekanan(Fd,opf,accident[0][1])
87          tempSigmaT=utama.SIGMA_T(p,tb,Tb)
88          sigmaT=tempSigmaT[0]
89
90          m0=utama.weibullParam(Tb,0)
91          phi12=utama.PHI(sigmaT,m0)
92          print('phiAwal='+str(phi12))
93          PHI12=[]
94          PHI12.append(phi12)
95          for i in range(1,len(accident)):
96                  Tuj=accident[i][1]          #temperatur pada ujung mesh, titik ke-i
97                  Taw=accident[i-1][1]        #temperatur pada awal mesh, titik ke (i-1)
98                  if i==1:
99                          Tm=(Tuj+Tb)/2
100                 else:
101                         Tm=(Tuj+Taw)/2
102
103                 taw=accident[i-1][0]
104                 tm=(accident[i-1][0]+accident[i][0])/2
105                 tuj=accident[i][0]
106
107                 dsaaw=utama.DS(Taw)
```

```
dsam1=utama.DS(Tm)
dsam2=utama.DS(Tm)
dsauj=utama.DS(Tuj)

tauAaw=dsaaw*taw
if tauAaw==0:
        Fdaw=utama.FTau(tauI)
else:
        Fdaw=(((tauI+tauAaw)*utama.FTau(tauI+tauAaw))-(tauAaw*utama.FTau(tauAaw)))/tauI

tauAm1=dsam1*taw
if tauAm1==0:
        Fdm1=utama.FTau(tauI)
else:
        Fdm1=(((tauI+tauAm1)*utama.FTau(tauI+tauAm1))-(tauAm1*utama.FTau(tauAm1)))/tauI

tauAm2=dsam2*tuj
if tauAm2==0:
        Fdm2=utama.FTau(tauI)
else:
        Fdm2=(((tauI+tauAm2)*utama.FTau(tauI+tauAm2))-(tauAm2*utama.FTau(tauAm2)))/tauI
        tauAuj=dsauj*tuj

if tauAuj==0:
        Fduj=utama.FTau(tauI)
else:
        Fduj=(((tauI+tauAuj)*utama.FTau(tauI+tauAuj))-(tauAuj*utama.FTau(tauAuj)))/tauI

opfaaw=utama.OPFAccident(Taw)
opfam1=utama.OPFAccident(Tm)
opfam2=utama.OPFAccident(Tm)
opfauj=utama.OPFAccident(Tuj)

paw=utama.tekanan(Fdaw,opfaaw,Taw)
pm1=utama.tekanan(Fdm1,opfam1,Tm)
pm2=utama.tekanan(Fdm2,opfam2,Tm)
puj=utama.tekanan(Fduj,opfauj,Tuj)

tempSigmaTaw=utama.SIGMA_T(paw,taw,Taw)
sigmaTaw=tempSigmaTaw[0]
tempSigmaTm1=utama.SIGMA_T(pm1,tm,Tm)
```

```
149                     sigmaTm1=tempSigmaTm1[0]
150                     tempSigmaTm2=utama.SIGMA_T(pm2,tm,Tm)
151                     sigmaTm2=tempSigmaTm2[0]
152                     tempSigmaTuj=utama.SIGMA_T(puj,tuj,Tuj)
153                     sigmaTuj=tempSigmaTuj[0]
154
155                     maw=utama.weibullParam(Taw,taw)
156                     phiaw=utama.PHI(sigmaTaw, maw)
157
158                     mm1=utama.weibullParam(Tm,tm)
159                     phim1=utama.PHI(sigmaTm1, mm1)
160
161                     mm2=utama.weibullParam(Tm,tm)
162                     phim2=utama.PHI(sigmaTm2, mm2)
163
164                     muj=utama.weibullParam(Tuj,tuj)
165                     phiuj=utama.PHI(sigmaTuj, muj)
166                     """batas update per tanggal 14-02-2018"""
167
168                     if i==1:
169                             selisih=phiaw-phi12
170                             if selisih>0:
171                                     phi12=phi12+selisih
172                             PHI12.append(phi12)
173                     else:
174                             selisih=phim2-phim1
175                             if (selisih)>0:
176                                     phi12=phi12+(phim2-phim1)
177                             PHI12.append(phi12)
178
179             del utama
180             rekap.close()
181             return PHI12
182
183     if __name__=="__main__":
184             if len(sys.argv)==4:
185                     f=sys.argv[1]
186                     statusLHS=int(sys.argv[2])
187                     temperature=sys.argv[3]
188                     """
189                     Eksekusi dengan 2 argumen, file input & opsi sampling: 0=tanpa sampling, 1=LHS, 2=SRS
```

```
190             """
191     else:
192             print("Argumen salah...")
193             exit(1)
194
195     x=readdata(f)
196     dimensi=x[0]
197     print('Len dimensi='+str(len(dimensi)))
198     uncertainties=x[1]
199     print('Len uncertainties='+str(len(uncertainties)))
200     characteristics=x[2]
201     print('Len character='+str(len(characteristics)))
202     irradiation=x[3]
203     print('Len irradiation='+str(len(irradiation)))
204     accident=x[4]
205     print('Len accident='+str(len(accident)))
206     rentang=x[5]
207     sampling=''
208
209     if statusLHS==0:
210             print("Tanpa random sampling")
211             phi=triacCalculation(dimensi,characteristics,irradiation,accident)
212             namafile='noSamplingRDE'+temperature
213             fnominal=file(namafile,'w')
214             for i in range(len(phi)):
215                     fnominal.write(str(accident[i][0]/(3600))+' '+str(phi[i])+'\n')
216             fnominal.close()
217     else:
218             if statusLHS==1:
219                     print('LHS')
220                     sampling='LHS'
221             elif statusLHS==2:
222                     print('SRS')
223                     sampling='SRS'
224             a=LHSCalculation(uncertainties,statusLHS)
225             samples=a.getSamples()
226             OPyCthickness=dimensi[0]-dimensi[1]
227             a.setOPyCNominal(OPyCthickness)
228             SiCthikness=dimensi[1]-dimensi[2]
229             a.setSiCNominal(SiCthikness)
230             IPyCNominal=dimensi[2]-dimensi[3]
```

```
231         a.setIPyCNominal(IPyCNominal)
232         bufferNominal=dimensi[3]-dimensi[4]
233         a.setBufferNominal(bufferNominal)
234         a.setKernelNominal(dimensi[4])
235         a.lhsCalculation()
236         kernel=a.getkernelListX()
237         print(np.var(kernel))
238
239         buff=a.getbufferListX()
240         print(np.var(buff))
241
242         ipyc=a.getIPyCListX()
243         print(np.var(ipyc))
244
245         sic=a.getSiCListX()
246         print(np.var(sic))
247
248         opyc=a.getOPyCListX()
249         print(np.var(opyc))
250
251         """fphiMin=file('phiMin','w')
252         fphiMax=file('phiMax','w')
253         phiMin=[]
254         phiMax=[]
255         phiRerata=[]
256         phiR=file('phiReRataTriangleLHS','w')"""
257
258         dist=a.getDist()
259         namafile=sampling+dist+str(samples)
260         fout=file(namafile,'w')
261         for i in range(samples):
262                 dimensi[5]=0.0
263                 dimensi[4]=kernel[i]+dimensi[5]
264                 dimensi[3]=buff[i]+dimensi[4]
265                 dimensi[2]=ipyc[i]+dimensi[3]
266                 dimensi[1]=sic[i]+dimensi[2]
267                 dimensi[0]=opyc[i]+dimensi[1]
268
269                 phi=triacCalculation(dimensi,characteristics,irradiation,accident)
270
271                 print('Sample ke-'+str(i)+', len(phi)='+str(len(phi)))
```

```
                for j in range(len(phi)):
                        if j!=len(phi)-1:
                                fout.write(str(phi[j])+' ')
                        else:
                                fout.write(str(phi[j])+'\n')
                        """if i==0:
                                phiRerata.append(phi[j])
                                phiMin.append(phi[j])
                                phiMax.append(phi[j])
                        else:
                                phiRerata[j]=phiRerata[j]+phi[j]
                                if phiMin[j]>phi[j]:
                                        phiMin[j]=phi[j]
                                if phiMax[j]<phi[j]:
                                        phiMax[j]=phi[j]"""

        fout.close()

        """for i in range(len(phiRerata)):
                a=float(phiRerata[i]/samples)
                phiR.write(str(accident[i][0]/(3600))+' '+str(a)+' '+str(phiMin[i])+' '+str(phiMax[i])+'\n')
        phiR.close()

        for i in range(len(phiMin)):
                fphiMin.write(str(accident[i][0]/(3600))+' '+str(phiMin[i])+'\n')
        fphiMin.close()

        for i in range(len(phiMax)):
                fphiMax.write(str(accident[i][0]/(3600))+' '+str(phiMax[i])+'\n')
        fphiMax.close()"""
```

Lampiran 6: LHScalculation.py

Listing 5: LHScalculation.py

```
1  from triangle import Triangle
2  from uniform import Uniform
3  from normal import Normal
4
5  class LHSCalculation():
6          def __init__(self,x,jenis):
7                  a=x[0]
8                  self.OPyCUncertainty=a[0]
9                  self.OPyCDistribution=a[1]
10                 self.OPyCVar=a[2]
11                 self.OPyCNominal=0.0
12                 self.OPyCListX=[]
13                 a=x[1]
14                 self.SiCUncertainty=a[0]
15                 self.SiCDistribution=a[1]
16                 self.SiCVar=a[2]
17                 self.SiCNominal=0.0
18                 self.SiCListX=[]
19                 a=x[2]
20                 self.IPyCUncertainty=a[0]
21                 self.IPyCDistribution=a[1]
22                 self.IPyCVar=a[2]
23                 self.IPyCNominal=0.0
24                 self.IPyCListX=[]
25                 a=x[3]
```

```
            self.bufferUncertainty=a[0]
            self.bufferDistribution=a[1]
            self.bufferVar=a[2]
            self.bufferNominal=0.0
            self.bufferListX=[]
            a=x[4]
            self.kernelUncertainty=a[0]
            self.kernelDistribution=a[1]
            self.kernelVar=a[2]
            self.kernelNominal=0.0
            self.kernelListX=[]
            a=x[5]
            self.Samples=a[0]
            self.Sampling=jenis
            self.Distribusi=''

    def getDist(self):
            return self.Distribusi

    def setOPyCNominal(self,x):
            self.OPyCNominal=x

    def getOPyCNominal(self):
            return self.OPyCNominal

    def setSiCNominal(self,x):
            self.SiCNominal=x

    def getSiCNominal(self):
            return self.SiCNominal

    def setIPyCNominal(self,x):
            self.IPyCNominal=x

    def getIPyCNominal(self):
            return self.IPyCNominal

    def setBufferNominal(self,x):
            self.bufferNominal=x

    def getBufferNominal(self):
```

```
67              return self.bufferNominal
68
69      def setKernelNominal(self,x):
70              self.kernelNominal=x
71
72      def getKernelNominal(self):
73              return self.kernelNominal
74
75      def getOPyCListX(self):
76              return self.OPyCListX
77
78      def getSiCListX(self):
79              return self.SiCListX
80
81      def getIPyCListX(self):
82              return self.IPyCListX
83
84      def getbufferListX(self):
85              return self.bufferListX
86
87      def getkernelListX(self):
88              return self.kernelListX
89
90      def getSamples(self):
91              return self.Samples
92
93      def lhsCalculation(self):
94              a=self.Samples
95              if self.OPyCUncertainty!=0:
96                      b=self.OPyCNominal
97                      c=b-self.OPyCUncertainty
98                      d=b+self.OPyCUncertainty
99                      print(b,c,d)
100                     if self.OPyCDistribution==0:
101                             """Triangle distribution for OPyC"""
102                             self.Distribusi='Triangle'
103                             e=Triangle(a,self.Sampling,'OPyC')
104                             e.setMod(b)
105                             e.setMin(c)
106                             e.setMax(d)
107                             e.generatePm()
```

```
            elif self.OPyCDistribution==1:
                """Uniform distribution for OPyC"""
                self.Distribusi='Uniform'
                e=Uniform(a,self.Sampling,'OPyC')
                e.setMin(c)
                e.setMax(d)
                e.generatePm()
            if self.OPyCDistribution==2:
                """Normal distribution for OPyC"""
                self.Distribusi='Normal'
                e=Normal(a,self.Sampling,'OPyC')
                e.setMean(b)
                e.setMin(c)
                e.setMax(d)
                e.setVar(self.OPyCVar)
                e.generatePm()

            self.OPyCListX=e.getX()

        if self.SiCUncertainty!=0:
            b=self.SiCNominal
            c=b-self.SiCUncertainty
            d=b+self.SiCUncertainty
            print(b,c,d)
            if self.SiCDistribution==0:
                """Triangle distribution for SiC"""
                self.Distribusi='Triangle'
                e=Triangle(a,self.Sampling,'SiC')
                e.setMod(b)
                e.setMin(c)
                e.setMax(d)
                e.generatePm()
            elif self.SiCDistribution==1:
                """Uniform distribution for SiC"""
                self.Distribusi='Uniform'
                e=Uniform(a,self.Sampling,'SiC')
                e.setMin(c)
                e.setMax(d)
                e.generatePm()
            if self.SiCDistribution==2:
                """Normal distribution for SiC"""
```

```
149                     self.Distribusi='Normal'
150                     e=Normal(a,self.Sampling,'SiC')
151                     e.setMean(b)
152                     e.setMin(c)
153                     e.setMax(d)
154                     e.setVar(self.SiCVar)
155                     e.generatePm()
156 
157             self.SiCListX=e.getX()
158 
159     if self.IPyCUncertainty!=0:
160             b=self.IPyCNominal
161             c=b-self.IPyCUncertainty
162             d=b+self.IPyCUncertainty
163             print(b,c,d)
164             if self.IPyCDistribution==0:
165                     """Triangle distribution for IPyC"""
166                     self.Distribusi='Triangle'
167                     e=Triangle(a,self.Sampling,'IPyC')
168                     e.setMod(b)
169                     e.setMin(c)
170                     e.setMax(d)
171                     e.generatePm()
172             elif self.IPyCDistribution==1:
173                     """Uniform distribution for IPyC"""
174                     self.Distribusi='Uniform'
175                     e=Uniform(a,self.Sampling,'IPyC')
176                     e.setMin(c)
177                     e.setMax(d)
178                     e.generatePm()
179             if self.IPyCDistribution==2:
180                     """Normal distribution for IPyC"""
181                     self.Distribusi='Normal'
182                     e=Normal(a,self.Sampling,'IPyC')
183                     e.setMean(b)
184                     e.setMin(c)
185                     e.setMax(d)
186                     e.setVar(self.IPyCVar)
187                     e.generatePm()
188 
189             self.IPyCListX=e.getX()
```

```
190
191             if self.bufferUncertainty!=0:
192                     b=self.bufferNominal
193                     c=b-self.bufferUncertainty
194                     d=b+self.bufferUncertainty
195                     print(b,c,d)
196                     if self.bufferDistribution==0:
197                             """Triangle distribution for buffer"""
198                             self.Distribusi='Triangle'
199                             e=Triangle(a,self.Sampling,'Buffer')
200                             e.setMod(b)
201                             e.setMin(c)
202                             e.setMax(d)
203                             e.generatePm()
204                     elif self.bufferDistribution==1:
205                             """Uniform distribution for buffer"""
206                             self.Distribusi='Uniform'
207                             e=Uniform(a,self.Sampling,'Buffer')
208                             e.setMin(c)
209                             e.setMax(d)
210                             e.generatePm()
211                     if self.bufferDistribution==2:
212                             """Normal distribution for buffer"""
213                             self.Distribusi='Normal'
214                             e=Normal(a,self.Sampling,'Buffer')
215                             e.setMean(b)
216                             e.setMin(c)
217                             e.setMax(d)
218                             e.setVar(self.bufferVar)
219                             e.generatePm()
220
221                     self.bufferListX=e.getX()
222
223             if self.kernelUncertainty!=0:
224                     b=self.kernelNominal
225                     c=b-self.kernelUncertainty
226                     d=b+self.kernelUncertainty
227                     print(b,c,d)
228                     if self.kernelDistribution==0:
229                             """Triangle distribution for kernel"""
230                             self.Distribusi='Triangle'
```

```
231             e=Triangle(a,self.Sampling,'Kernel')
232             e.setMod(b)
233             e.setMin(c)
234             e.setMax(d)
235             e.generatePm()
236         elif self.kernelDistribution==1:
237             """Uniform distribution for kernel"""
238             self.Distribusi='Uniform'
239             e=Uniform(a,self.Sampling,'Kernel')
240             e.setMin(c)
241             e.setMax(d)
242             e.generatePm()
243         if self.kernelDistribution==2:
244             """Normal distribution for kernel"""
245             self.Distribusi='Normal'
246             e=Normal(a,self.Sampling,'Kernel')
247             e.setMean(b)
248             e.setMin(c)
249             e.setMax(d)
250             e.setVar(self.kernelVar)
251             e.generatePm()
252
253         self.kernelListX=e.getX()
```

Lampiran 7: lhs.py

Listing 6: lhs.py

```
1  import math
2  from matplotlib import pyplot as plt
3  import numpy as np
4  class triacLHS():
5          def __init__(self,n,jenis,param):
6                  self.jmlSample=n
7                  self.Jenis=jenis
8                  """1=LHS; 2=SRS"""
9                  self.Pm=[]
10                 self.X=[]
11                 self.Zone=0
12                 self.PDF=[]
13                 self.CDF=[]
14                 self.Min=0.0
15                 self.Mod=0.0
16                 self.Mean=0.0
17                 self.Max=0.0
18                 self.StdDev=0.0
19                 self.Var=0.0
20                 self.Parameter=param
21                 self.Dist=''
22
23         def generatePDF(self):
24                 """counts,bin_edges=np.histogram(self.X,bins=a)
25                 b=counts.tolist()"""
```

```
            if self.Jenis==1:
                    fout=file('PDF_LHS'+'_'+self.Dist+'_'+self.Parameter,'w')
            elif self.Jenis==2:
                    fout=file('PDF_SRS'+'_'+self.Dist+'_'+self.Parameter,'w')
            """for i in range(a):
                    fout.write(str(i+1)+' '+str(b[i])+'\n')"""
            if self.Dist=='Normal':
                    a=len(self.X)
                    self.Min=min(self.X)
                    self.Max=max(self.X)
                    print(self.Min, self.Max)
            Interval=(self.Max-self.Min)/self.Zone
            bins=[0 for i in range(self.Zone)]

            ftemp=file('normalPDFtemp','w')
            for i in range(self.Zone):
                    m0=(i*Interval)+self.Min
                    m1=((i+1)*Interval)+self.Min
                    for j in range(len(self.X)):
                            ftemp.write(str(m0)+' '+str(self.X[j])+' '+str(m1)+'\n')
                            if self.X[j]>m0 and self.X[j]<=m1:
                                    bins[i]=bins[i]+1
                    fout.write(str(i+1)+' '+str(bins[i])+'\n')
            fout.close()
            ftemp.close()

    def setMin(self,m):
            self.Min=m

    def setMod(self,m):
            self.Mod=m

    def setMean(self,m):
            self.Mean=m

    def setMax(self,m):
            self.Max=m

    def setStdDev(self,s):
            self.StdDev=s

```

```
67          def setVar(self,v):
68                  self.Var=v
69
70          def setZone(self,z):
71                  self.Zone=z
72
73          def getPm(self):
74                  a=self.jmlSample
75                  fout=file('uniformPm','w')
76                  for i in range(a):
77                          """print self.Pm[i]"""
78                          fout.write(str(i+1)+' '+str(self.Pm[i])+'\n')
79                  fout.close()
80
81          def getX(self):
82                  return self.X
```

Lampiran 8: uniform.py

Listing 7: uniform.py

```
from lhs import triacLHS
import random
import math
class Uniform(triacLHS):
        def __init__(self,n,jenis,params):
                triacLHS.__init__(self,n,jenis,params)
                self.Dist='Uniform'

        def generatePm(self):
                random.seed()
                if self.Jenis==1:
                        fout=file('X_LHS'+'_'+self.Dist+'_'+self.Parameter,'w')
                elif self.Jenis==2:
                        fout=file('X_SRS'+'_'+self.Dist+'_'+self.Parameter,'w')
                for i in range(self.jmlSample):
                        r=random.random()
                        if self.Jenis==1:
                                a=(float)(r+i)/(self.jmlSample)
                                self.Pm.append(a)
                        elif self.Jenis==2:
                                a=r
                        x=(a*(self.Max-self.Min))+self.Min
                        self.X.append(x)
                random.shuffle(self.X)

```

```
26                  for i in range(self.jmlSample):
27                          fout.write(str(i+1)+' '+str(self.X[i])+'\n')
28                  fout.close()
```

Lampiran 9: triangle.py

Listing 8: triangle.py

```
from lhs import triacLHS
import random
import math
class Triangle(triacLHS):
        def __init__(self,n,jenis,params):
                triacLHS.__init__(self,n,jenis,params)
                self.Dist='Triangle'

        def generatePm(self):
                if self.Min == self.Mod:
                        k=0.0
                elif self.Mod == self.Max:
                        k=1.0
                elif self.Min!=self.Mod and self.Mod!=self.Max:
                        k=(float)(self.Mod-self.Min)/(self.Max-self.Min)
                random.seed()
                if self.Jenis==1:
                        fout=file('X_LHS'+'_'+self.Dist+'_'+self.Parameter,'w')
                elif self.Jenis==2:
                        fout=file('X_SRS'+'_'+self.Dist+'_'+self.Parameter,'w')
                for i in range(self.jmlSample):
                        r=random.random()
                        if self.Jenis==1:
                                a=(float)(r+i)/(self.jmlSample)
                                self.Pm.append(a)
```

```
26                  elif self.Jenis==2:
27                          a=r
28
29                  if a<=k:
30                          x=self.Min+math.sqrt(a*(self.Max-self.Min)*(self.Mod-self.Min))
31                          self.X.append(x)
32                  if a>k:
33                          x=self.Max-math.sqrt((1-a)*(self.Max-self.Min)*(self.Max-self.Mod))
34                          self.X.append(x)
35
36          random.shuffle(self.X)
37
38          for i in range(self.jmlSample):
39                  fout.write(str(i+1)+' '+str(self.X[i])+'\n')
40          fout.close()
```

Lampiran 10: normal.py

Listing 9: normal.py

```
from lhs import triacLHS
import random
import math
class Normal(triacLHS):
        def __init__(self,n,jenis,params):
                triacLHS.__init__(self,n,jenis,params)
                self.Dist='Normal'

        def generatePm(self):
                random.seed()
                if self.Jenis==1:
                        fout=file('X_LHS'+'_'+self.Dist+'_'+self.Parameter,'w')
                elif self.Jenis==2:
                        fout=file('X_SRS'+'_'+self.Dist+'_'+self.Parameter,'w')
                for i in range(self.jmlSample):
                        r=random.random()
                        if self.Jenis==1:
                                a=(float)(r+i)/(self.jmlSample)
                                self.Pm.append(a)
                        elif self.Jenis==2:
                                a=r
                        x=(self.Var*math.sqrt(2)*((2*a)-1))+self.Mean
                        """x=(self.Mean*math.sqrt(2)*((2*a)-1)+self.Var)"""
                        self.X.append(x)
                random.shuffle(self.X)
```

```
26
27              for i in range(self.jmlSample):
28                      fout.write(str(i+1)+' '+str(self.X[i])+'\n')
29              fout.close()
30              """
31              Supaya agak mirip ekspektasi, minimum dan maksimum harus didefinisikan secara manual.
32              Konsekuensinya, harus disesuaikan dengan nilai variance. Semakin besar variance, rentang nilai minimum dan maksimum
33              bisa terlampaui.
34              """
```