PWRinSPACE
Symulator silnika na paliwo stałe dla geometrii BYTES

Manfred Gawlas

29.11.2023

Abstract

Dokument ten przedstawia ręcznie napisany symulator silnika na paliwo stałe dla różnych typów pochodnych od geometrii BYTES, algortym wykonujący symulacje oraz prezentuje wzory fizyczne potrzebne do przeprowadzenia takiej symulacji.

1 Input

- Dane dla konkretnego paliwa: $\rho_p, T_0, k, R_s, a, n$
- Burn rate coefficient a
- Pressure exponent n
- Funkcje Area burning $A_b(t, r)$
- Początkowe ciśnienie w komorze P_0
- Rozmiary komory oraz rdzeniaL, D, d
- Ciąg początkowy przed poprawką F_{pocz}

2 Output

- Ciąg F
- Wymiary dyszy: A_t, A_e
- Czas spalania t
- Impuls całkowity I_t

3 Wzory

Wszystkie potrzebne w tym symulatorze wzory pochodzą z "Rocketry formulas and derivations" autorstwa Sebastiana Króla(BAZA).

1) Pierwszym ważnym wzorem którego będziemy używać jest wzór na ciąg silniku.

$$F = A_t P_0 \sqrt{\frac{2k^2}{k-1}\left(\frac{2}{k+1}\right)^{\frac{k+1}{k-1}}\left[1-\left(\frac{P_e}{P_0}\right)^{\frac{k-1}{k}}\right]} \tag{1}$$

Zakładając C_F równe

$$C_F = \sqrt{\frac{2k^2}{k-1}\left(\frac{2}{k+1}\right)^{\frac{k+1}{k-1}}\left[1-\left(\frac{P_e}{P_0}\right)^{\frac{k-1}{k}}\right]} \tag{2}$$

Otrzymujemy

$$F = A_t P_0 C_F \tag{3}$$

2) Jako że mamy podany ciąg początkowy F_0 i zakładamy sobie jakieś ciśnienie początkowe to liczymy dla nich A_t.

$$A_t = \frac{F_0}{P_0 C_F} \tag{4}$$

3) Dla A_e obliczamy je z

$$A_e = \frac{A_t}{\left(\frac{k+1}{2}\right)^{\frac{1}{k-1}}\left(\frac{P_e}{P_0}\right)^{\frac{1}{k}}\sqrt{\left(\frac{k+1}{k-1}\right)\left[1-\left(\frac{P_e}{P_0}\right)^{\frac{k-1}{k}}\right]}} \tag{5}$$

4) Ścisły wzór na impuls całkowity

$$Ic = \int_0^{t_1} F(t)dt$$

Całke tą będziemy rozwiązywać metodą graficzną prostokontów, ale dokładniej o tym w sekcji o algorytmie.

5) Ciśnienie jako funkcja A_b:

$$P_{ch}(A_b) = K_n^{\frac{1}{1-n}}(c^* \rho_p a)^{\frac{1}{1-n}} \tag{6}$$

Co wyprowadzamy z wzorów:

$$r = ap^n$$

$$\dot{m} = r\rho_p A_b$$

$$c^* = \frac{A_t P_{ch}}{\dot{m}}$$

4 Geometria ziarna

Projekt ten będzie brał pod uwagę tylko geometrie typu BYTES, jako że inne zdają sie albo nie spełniać wymagać albo są ciężkie do policzenia. Geometrie typu BYTES można zaposać w ogólnej postaci:

$$A_b(t) = A_w(t) + kA_z(t)$$

gdzie k to liczba powierzchni bocznych walca które ulegają spalaniu. Dla 1 walcowego modelu jest to liczba z przedziału $k \in \{0, 1, 2\}$. Dla geometrii bytes z wieloma walcami dochodzą nam dodatkowe powierzchnie.

$$A_w(t) = 2\pi R(t)L(t)$$

gdzie A_w powierzchnia spalania wewnętrzna(zewnętrzna rdzenia), R promień rdzenia, L suma długość rdzeni.

$$A_z(t) = \pi\frac{D^2}{4} - \pi R^2(t)$$

gdzie A_z powierzchnia spalania na jednej podstawie walca, D średnica komory spalania, R promień rdzenia. To jest zakładając że rdzenie wszystkich rdzeni jest tożsamy.

Funckcje $L(t)$ oraz $R(t)$ wyrażają sie wzorami, lub przez ciągi rekurencyjne:

$$L(t) = L_0 - kr(P_0)t$$
$$L_n = L_{n-1} - kr_n\Delta t$$
$$R(t) = \frac{d}{2} + r(P_0)t$$
$$R_n = R_{n-1} + r_n\Delta t$$

5 Algorytm

W tej sekcji przedstawie po krótce jak działa główny algorytm symulatora silniku. Jest on częściowo tożsamy z algorytmem liczącym całke oznaczoną metodą prostokątków. Impuls całkowiny jest liczony właśnie całką dla czasu Δt oraz ciągu chwilowego F_0.

$$Ic \approx \sum_{i=0}^{n} F_i\Delta t$$

Gdzie Δt to dowolny mały okres czasu dla którego przyjmujemy $P_0 = const$, $r = const$. Co za tym idzie zakładamy $F_i = const$ dla tego okresu. Kolejne F_i będą wyznaczane podczas działania algorytmu.

Pierwszym krokiem jest policznenie poprawki dla założonego ciśnienia i ciągu początkowego. Kożystamy wiec z równania (4) i wyznaczamy A_t. Następnie przy pomocy wzoru (6) na P_ch wyznaczamy rzeczywiste ciśnienie. Po policzeniu poprawki możemy zacząć główną pętle programu.

Warunek pętli

Pętla wykonuje się dopuki nie spali się cale paliwo, a więc dopuki zmienna

$$\text{totalRegressed} < \frac{D-d}{2}$$

Krok 1

Liczymy chwilowy ciąg silnika oraz następnie dodajemy jego iloczyn(całka metodą prostokątu) do impulsu.

$$F = A_t P_0 C_F$$

$$Ic+ = F\Delta t$$

Krok 2

Liczymy chwilową regresje dla tego okresu czasu:

$$r = aP_0^n$$

Krok 3

Z pomocą obliczonej regresji obliczamy jaka zmiana zaszła dla wartości A_b

$$R = R + r\Delta t$$

$$L = L - kr\Delta t$$

Krok 4

Obliczamy nową wartość P_0 za pomocą wzoru (6)

$$P_0(A_b) = K_n^{\frac{1}{1-n}} (c^* \rho_p a)^{\frac{1}{1-n}}$$

Krok 5

Aktualizujemy zmienną totalRegressed.

$$\text{totalRegressed}+ = \Delta tr$$

6 Symulator

W pierwszej części tej sekcji umieszcze jego implementacje w C, a w drugiej przedstawie porównanie dla modelu 32x120 mm, $d = 20mm$, zakładane ciśnienie $P_0 = 30atm$, w geometrii BYTES, z 1 powierzchnią podstawy palenia, pomiędzy moim symulatorem a programem openRocket.

Implementacja w C

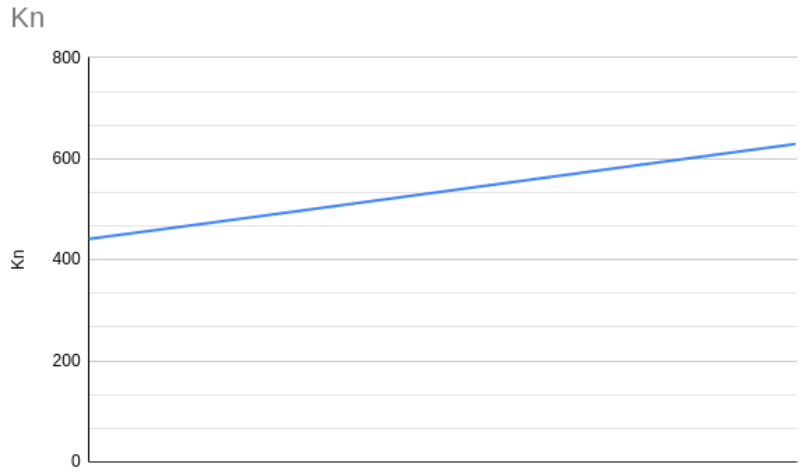
```
// ster.c Manfred Gawlas

#include <stdio.h>
#include <math.h>

struct AbGeometry // Grain size
{
  double L;
  double R;
  double D;
};

struct General // General variables that we often use
{
  double Pch;
  double L;
  double D;
  double d;
  double At;
  double Pe;
};

struct Fuel // Size of grain
{
  double Cstar;
  double a;
  double n;
  double k;
  double rho;
};

double CF(struct General Cylinder, struct Fuel RNX71) // Function
    that returns value of CF
{
  return pow((2*RNX71.k*RNX71.k)/(RNX71.k-1) * pow(2/(RNX71.k+1), (
    RNX71.k+1)/(RNX71.k-1)) * (1 - pow(Cylinder.Pe/Cylinder.Pch, (
    RNX71.k-1)/RNX71.k)), 0.5);
}

double F(struct General Cylinder, struct Fuel RNX71) // Function
    that returns thursh value
{
  return CF(Cylinder, RNX71) * Cylinder.At * Cylinder.Pch;
}

```

```
42 double FunctionAb(struct AbGeometry Ab) // Function that returns
       area burning
43 {
44   return 3.14159*(2*Ab.L*Ab.R + 0.25*Ab.D*Ab.D - Ab.R*Ab.R);
45 }
46
47
48 int main(void)
49 {
50   // All values are in basic SI units
51
52   double r=0;
53   double Ic=0;
54   double F0=82.37528;
55   double tc=0;
56   double totalRegressed=0; // Stores how deap it burned in
57   double Deltat=0.003; // Time period for which our loop works
58
59   struct General Cylinder;
60   Cylinder.Pch=30*101325;
61   Cylinder.L=0.12;
62   Cylinder.D=0.032;
63   Cylinder.d=0.020;
64   //Cylinder.At=0.00001824875;
65   Cylinder.Pe=101325;
66
67   struct Fuel RNX71;
68   RNX71.Cstar=779;
69   RNX71.a=0.0000163938; // Counted by hand from a for MPa, this one
       is for Pa
70   RNX71.n=0.371;
71   RNX71.k=1.18;
72   RNX71.rho=1848;
73
74   struct AbGeometry Ab;
75   Ab.L=Cylinder.L;
76   Ab.R=Cylinder.d/2;
77   Ab.D=Cylinder.D;
78
79   //printf("F0=%f\n", F(Cylinder, RNX71));
80   //printf("CF=%f\n\n", CF(Cylinder, RNX71));
81
82
83   Cylinder.At=F0/(Cylinder.Pch*CF(Cylinder, RNX71));
84   Cylinder.Pch=pow(FunctionAb(Ab) / Cylinder.At, 1/(1-RNX71.n)) *
       pow(RNX71.Cstar * RNX71.rho * RNX71.a, 1/(1-RNX71.n));
85
86
87   ////////////////////////////////
88   // Main loop, algorithm similar to calculating integrad by
       rectangles.
89   ////////////////////////////////
90
91   while(totalRegressed<((Cylinder.D-Cylinder.d)/2))
92   {
93     //  Value of time, thrust and updating value of impuls
94
```

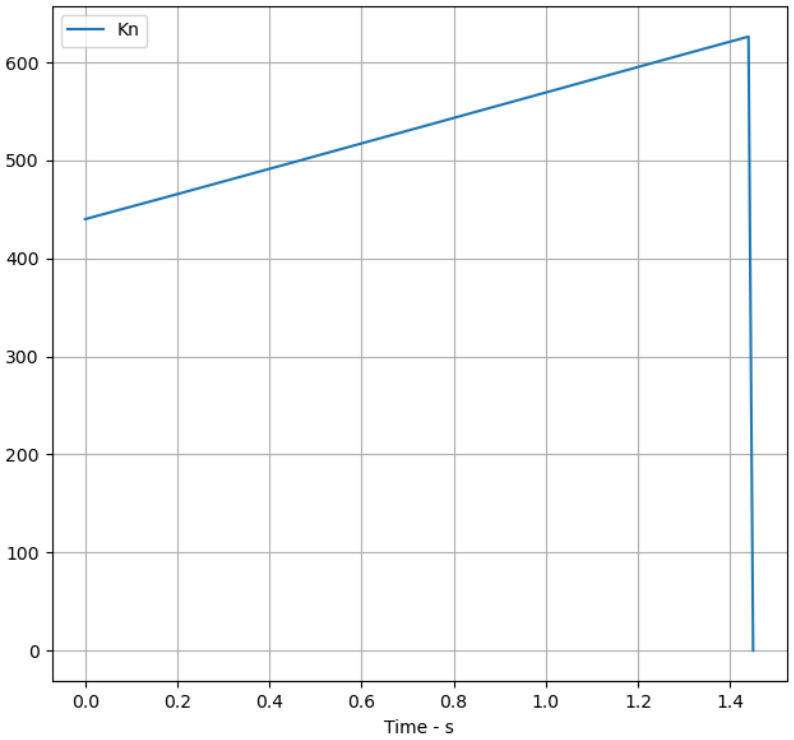
```
95      tc+=Deltat;
96      F0=F(Cylinder, RNX71);
97      Ic+=F0 * Deltat;
98
99
100     ////////////////////////////////
101     // You can print values for any variable. Helps with making
        graphs.
102     ////////////////////////////////
103
104     printf("%f\n", F0);
105     //printf("%f\n\n", Ic);
106     //printf("%f\n", FunctionAb(Ab)/Cylinder.At); // Kn
107     //printf("%f\n", Cylinder.Pch);
108
109     ////////////////////////////////
110
111     r=RNX71.a * pow(Cylinder.Pch, RNX71.n);
112     Ab.L=Ab.L - r*Deltat;
113     Ab.R=Ab.R + r*Deltat;
114
115     Cylinder.Pch=pow(FunctionAb(Ab) / Cylinder.At, 1/(1-RNX71.n)) *
         pow(RNX71.Cstar * RNX71.rho * RNX71.a, 1/(1-RNX71.n));
116
117     totalRegressed+=r*Deltat; // Updates how much already burned
118   }
119
120
121   // Print of end values
122
123   printf("Ic=%f\n", Ic);
124   printf("tc=%f\n", tc);
125   printf("Pch=%f\n", Cylinder.Pch);
126
127   return 0;
128 }
```

Porównanie symulatorów

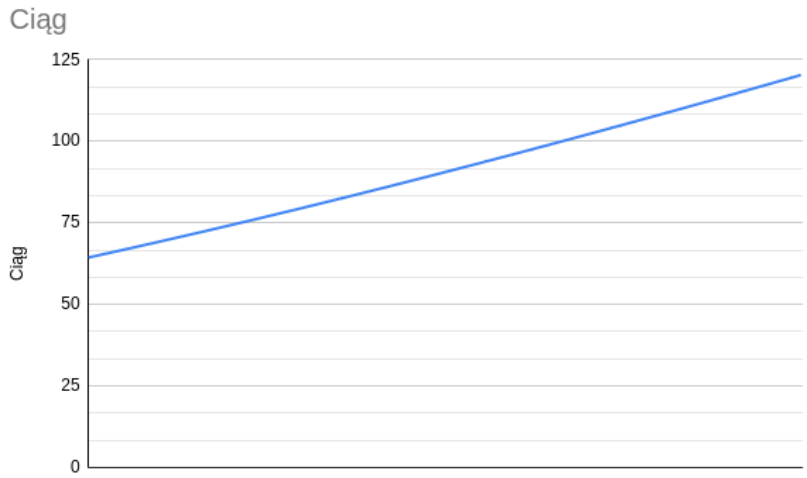
Wykresy K_n.
Wykres 1 mój symulator:



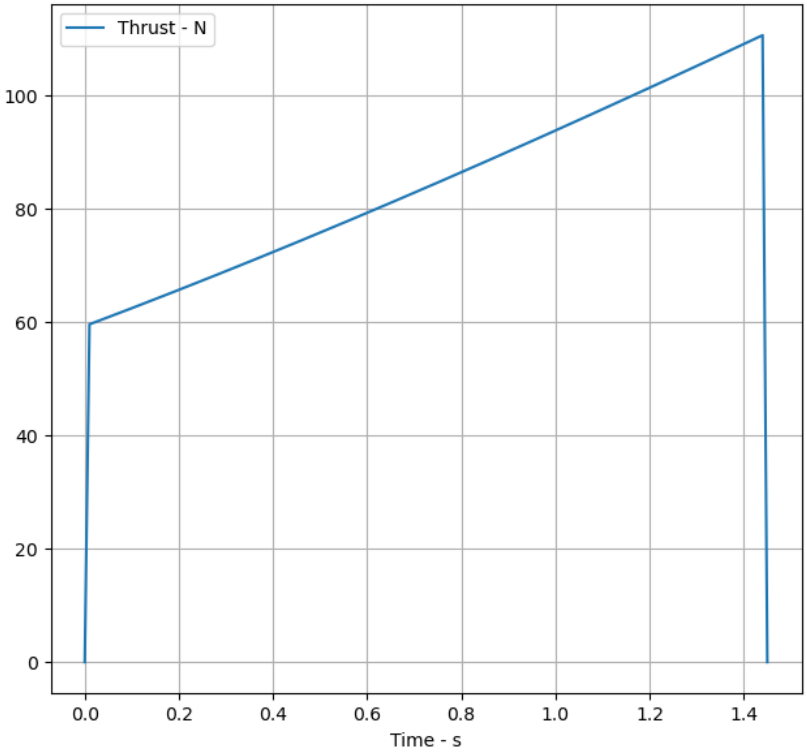
Wykres 2 openMotor:



Wykresy ciągu.
Wykres 1 mój symulator:



Wykres 2 openMotor:



Podsumowując, symulatory zwracają bardzo podobne wyniki, gdzie błąd jest spowodowany w dużym stopniu przez różne przybliżenia dla niektórych parametrów. Np. fakt że do openMotora można wpisać współczynnik regresji tylko dla 2 miejsc znaczących, generuje 4% błędu na ciśnieniu itd. Wiadomo że dane te są i tak niedokładne, więc te kilku procentowe błędy ostatecznie raczej nie są do końca możliwe do pozbycia się w sensownym stopniu.