Programowanie jako narzędzie inżyniera XXIw

06.12.2024 | Tomasz Tomanek | ABB |ELSP | LVS | tomasz.tomanek@pl.abb.com

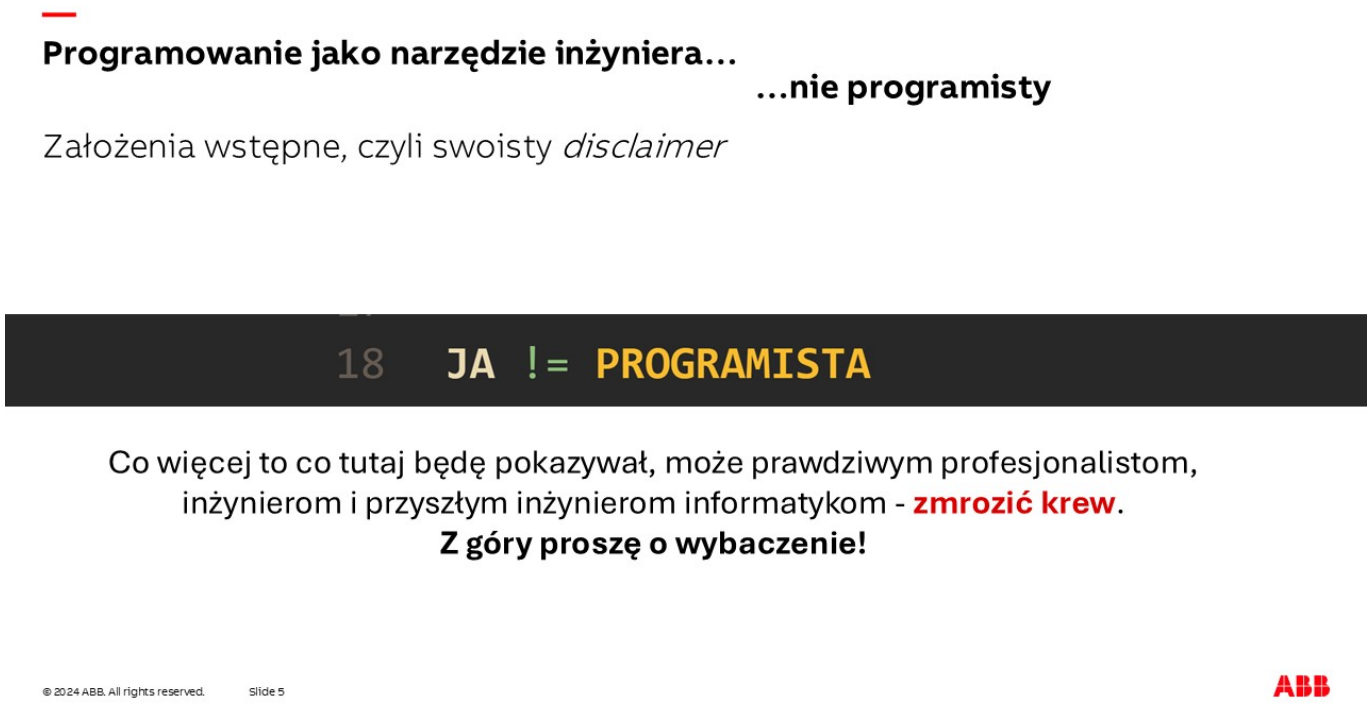
Ta prezentacja w swoich założeniach nie jest zaplanowana jako wykład, który miałby czegoś szczególnego czy konkretnego uczyć. Spodziewam się nawet, że wielu z Was wie na poruszane tutaj tematy niewspółmiernie więcej niż ja.

O czym zatem chciałbym nieco opowiedzieć?

O wykorzystaniu programowania jako narzędzia w codziennej pracy inżyniera. Inżyniera nie programisty, nie związanego profesjonalnie z **informatyką**.

Na wstępie, musze się do czegoś przyznać i coś wyraźnie powiedzieć: *Nie jestem programistą, informatykiem czy specjalistą w żadnej pokrewnej dziedzinie*

Jestem inżynierem elektrykiem zajmującym się zawodowo zagadnieniami związanymi z przemysłowymi rozdzielnicami elektrycznymi niskiego napięcia. Ich projektowaniem, badaniami i rozwojem.



W zwiazku z powyższym, przedstawione dalej informacje i podejście nie koniecznie jest zgodne z kanonem realizacji zagadnień związanych z programowaniem jaki przyjęty jest właśnie we wspomnianych dziedzinach ogólnie pojętej *informatyki*.

Innymi słowy to co tu będę poruszał, może prawdziwym profesjonalistom z tej dziedziny - zmrozić krew. Za co z góry przepraszam.

O czym zatem mam zamiar mówić?

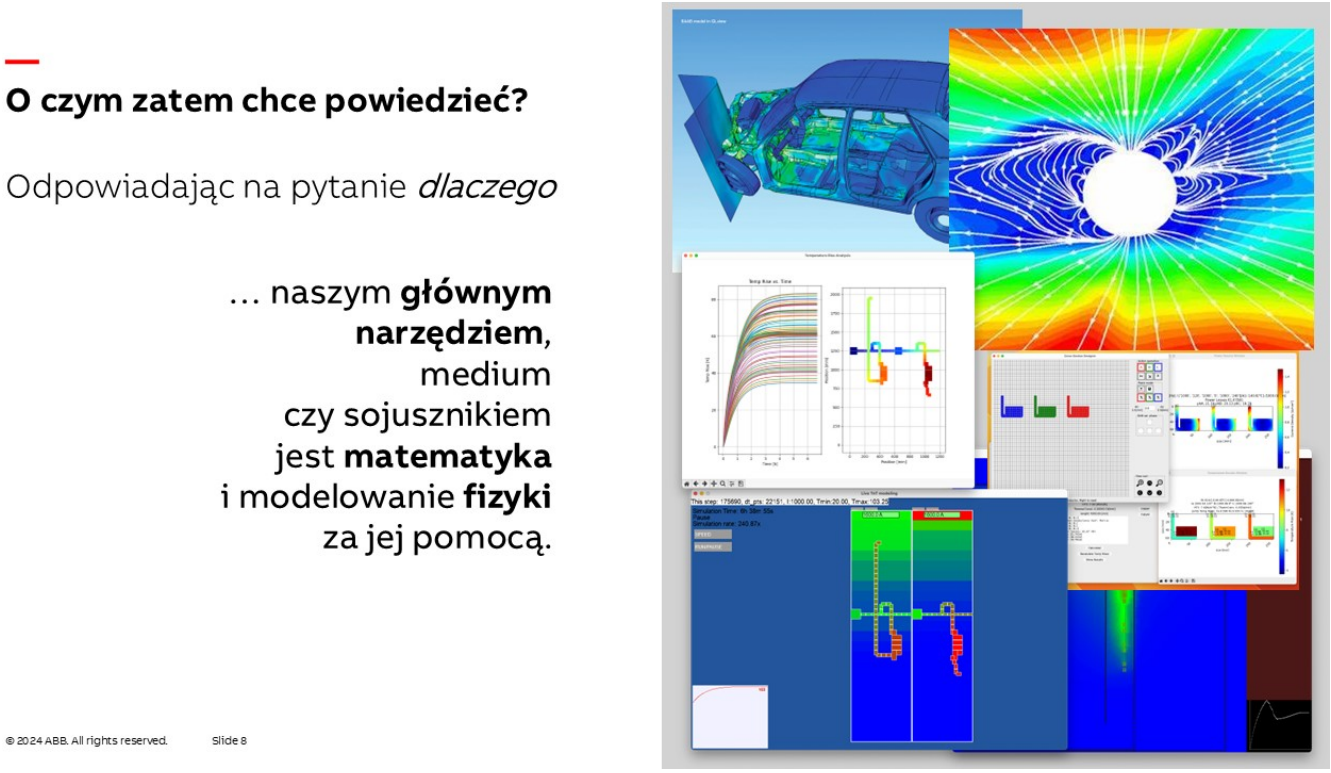
O tym, że codzienna praca inżyniera związana jest z ciągłym podejmowaniem technicznych decyzji. Wybieraniem sposobu realizacji danego rozwiązania. Jak kiedys powiedział mi pewien starszy inżynier konstruktor z wieloletnim doświadczeniem - projektowanie rozwiązań inżynieryjnych składa się, z szeregu małych, codziennych wynalazków.

Spotkałem się też gdzieś z cytatem, że

> Praca inżyniera różni się od pracy naukowca tym, że naukowcy odpowiadają na pytanie **dlaczego** tak się dzieje, a inżynierowie na pytanie **jak** to zrobić.

Moja zaś teza, którą stawiam po tych już prawie 20 latach zawodowych doświadczeń, jest taka iż - warto wybierając odpowiedź na pytanie "jak?" wiedzieć "dlaczego" właśnie ją wybieramy.

Inżynierskie odpowiadanie na pytanie dlaczego



Zatem dobra odpowiedź na wspomniane pytanie *jak* niejako narzuca nam swoistą konieczność uzasadnienia tejże. Nasze inżynierskie decyzje winny byc podejmowane świadomie w oparciu o najlepsze zrozumienie zjawisk fizycznych z jakimi opracowywane rozwiązanie musi się zmierzyć.

W poszukiwaniu tego zrozumienia i tych odpowiedzi naszym narzędziem jest fizyka i jej główny język komunikacji - czyli matematyka.

Matematyka tak, dość szybko potrafi stać się nieco zawiła i nie taka oczywista do rozwikłania opisanych nią modeli.

I tutaj z pomocą przychodzą nam benefity tego, że żyjemy w **erze informacji** czyli mamy do dyspozycji potężnego sprzymierzeńca w postaci komputerów no i nieodzownego dla ich użyteczności - oprogramowania.

Komputery mogą wszystko?

Wikipedia w swoim artykule listuje prawie 50 pakietów oprogramowania pod hasłem *finite elements software packages*. Innymi słowy w szeroko pojętej domenie rozwiązań obliczeniowo symulacyjnych nie możemy narzekać na braki.

Spoglądając wstecz, dostrzegam także istotny rozwój tychże rozwiązań. Rozwój (patrząc na moje nie nazbyt obszerne doświadczenia z pakietem AnSys) nie tylko dotyczący samych sinków obliczeniowych czy mechanizmów umożliwiających nowe rodzaje badań symulacyjnych, ale także rozwój interfejsu użytkownika. Z biegiem *krzywa wejścia* w uzywanie niektórych z tych systemów coraz bardziej łagodnieje.

Ma to jak najbardziej wiele zalet. Jednak, ma to też swoje - specyficzne, może nie wady ale bardziej może nieść ze sobą pewne ryzyka.

Jakie? można zapytać. Takie związane z tym, że nader łatwa obsługa programu może uniewrażliwić nowych użytkowników na konieczność dopilnowania jakości danych wejściowych do takich analiz. Analizy FEMowskie, dla przykładu z rodziny CFD mają taki charakter, że nie jest to jednoznaczne rozwiązanie zestawu równań. Nie ma tutaj jedynego słusznego wyniku, a raczej minimalizujemy błędy stopniowo prowadząc nasze rozwiązanie

stanu zbieżności. Ryzyko w tym, że wynik choć z punktu widzenia solwera już osiągnięty i numerycznie poprawny - nie oddaje rzeczywistości. Chociażby ze względu na niesłusznie przyjęte parametry brzegowe, startowe czy wewnątrz-symulacyjne.

Innymi słowy - mamy do czynienia z systemem SISO*. Jakość danych wejściowych determinuje jakość i poprawność uzyskanych wyników.

A dlaczego uważam, że ma to coś wspólnego z interfejsem użytkownika? Ot dlatego, że łatwość obsługi może zmniejszać ilość nauki i szkolenia, a to może prowadzić do niezrozumienia zasad działania naszego systemu analiz czy symulacji.

**rozwinięcie skrótu pozostawiam czytelnikowi*

Innymi słowy...

Z wielką mocą przychodzi wielka odpowiedzialność

To taka komiksowa maksyma wujka Bena. Ale ma tu na tyle zastosowanie, że korzystając z tak wyszukanych systemów - trzeba wiedzieć co się robi. Albo przynajmniej wiedzieć czego się spodziewać jako wyniku - jeszcze przed rozpoczęciem analiz.

Z wielką mocą przychodzi wielka odpowiedzialność

…czyli musimy wiedzieć kiedy narzędzia dają dobre odpowiedzi!

Poziom analizy	Preferowane podejście	Przykładowe narzędzia CAE
Ogólne rozważania wstępne	uproszczony model fizyczny, dający dobre pierwsze przybliżenie.	Kartka i ołówek
Interpolacja na podstawie istniejących danych testowych itp..	Podstawowe modele fizyczne, takie jak na przykład sieci statyczne termiczne, belki…	Narzędzia takie jak Excel itp..
Ekstrapolowanie parametrów istniejących rozwiązań poza przebadany zakres	Podejście Quasi-Symulacyjne z wykorzystaniem modeli iteracyjnych itp.	Symulatory Dynamiczne Sieci Termicznych TnT2.0, Iteracyjne *solvery* elektryczne, mechaniczne
Skomplikowane analizy porównawcze	Quasi-Symulacje – na przykład komórkowe (PGC, CSD)	PGC 2D/3D, CSD
Nowe rozwiązania, dalece idące modyfikacje dla krytycznych podsystemów	Zaawansowane rozwiązania FEM	Ansys, Fluent, ThermNet, Magnet….

Testy i badania

 Slide 10

Pomocną dla mnie w zapewnieniu takiego podejścia okazała się opracowana już dość dawno temu w ramach mojego zajmowania się w dziale R&D promowaniem świadomego i opartego na obliczeniach i analizach projektowania - metodyka *kolejnych przybliżeń*.

Metodyka kolejnych przybliżeń

Polega ona na tym aby każde zagadnienie nad jakim pracujemy, a które wymaga albo przynajmniej warto by było oprzeć o modele fizyczno-obliczeniowe, rozpatrywać w na kilku poziomach przybliżenia.

W **pierwszym** podejściu skupić się na najprostszym modelu fizycznym, takim pierwszy przyblizeniu. Modelu, który możemy obliczyć na *kawałku papieru*. Nie oczekując tutaj precyzyjnych wyników - a raczej zorientowania się, jakiego rzędu wielkości się spodziewamy i jaki jest spodziewany charakter odpowiedzi naszego analizowanego układu.

W kolejnych stopniach przybliżenia możemy korzystać z bardziej rozbudowanych modeli, czy równań. Bazować nasze analizy czy przybliżenia na interpolacji, a później i ekstrapolacji już istniejących i zweryfikowanych realnymi badaniami rozwiązań i ich właściwości.

Wraz ze wzrostem skomplikowania stojącego przed nami wyzwania możemy dojść do wykorzystania już wspomnianych profesjonalnych i zaawansowanych systemów symulacyjno-analitycznych.

Jednak mając już wiedze choćby z kroku **pierwszego** możemy wyniki tych symulacji zderzyć z naszymi wstępnymi oszacowaniami - zmniejszając w tem sposób ryzyko nie zauważenia znacznych błędów.

Ostateczną wersją weryfikacji w tak ujętym podejściu są zawsze fizyczne testy. Takie testy laboratoryjne lub inne tego typu - gdzie możemy zweryfikować rzeczywiste właściwości systemu.

XXIw to era dostępu do narzędzi

Praktycznie każdy z nas ma już dostępu do jakiegoś środowiska, w którym możemy zrealizować nasze obliczeniowe potrzeby.

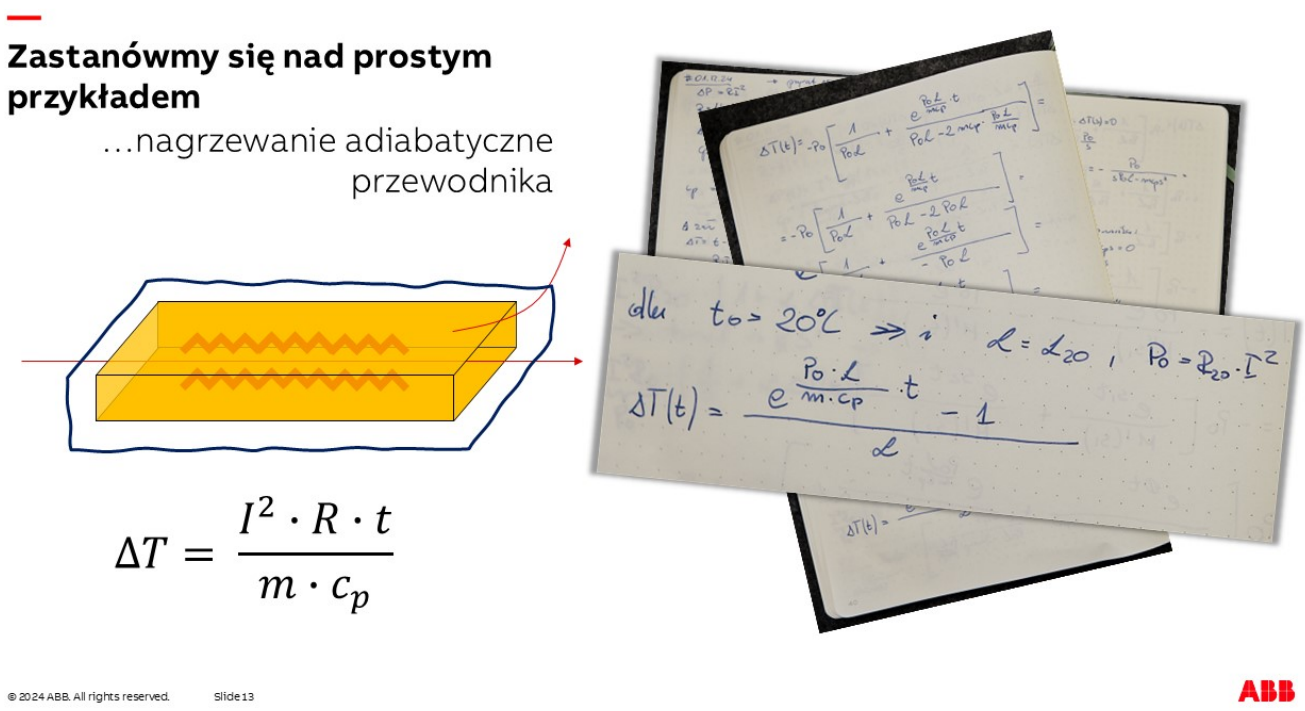
Napisać skrypt, czy program taki czy inny, który ułatwi nam nieco Trudy codzienności inżynierskiej.

BA!

Niewykluczone, że już to zrobiliśmy – czasem nawet nieświadomie!

Przykład

Starczy już nam może tych dywagacji. Spójrzmy zatem na przykład z życia wzięty.



Przeprowadźmy takie rozważania w dziedzinie elektryczno-termicznej. Rozważania mające na calu odpowiedź na pytanie - jak zmieni się temperatura przewodnika o znanym materiale i rozmiarze ($\sigma, \alpha, a,b,l, c_{p}, \rho$) gdy przepuścimy przezeń prąd elektryczny o znanym natężeniu I przez znany czas τ.

W tej analizie załóżmy, że nasz przewodnik nie oddaja ciepła do otoczenia - czyli rozważamy przypadek adiabatyczny. Nie jest to założenie czysto akademickie, gdyż dla niewielkich czasów takiej analizy (do powiedzmy 3s) jest to założenie wystarczająco zbieżne z rzeczywistością, i dopuszczone przez normy.

Jako, że nasz przewodnik nie jest idealnym, i posiada swoją rezystancję, określoną tutaj przez jego przewodność właściwą σ oraz jego przekrój $A=a\cdot b$ i długość l:

$$ R = \frac{l}{a\cdot b\cdot \sigma} [\Omega] $$

wiemy, że pojawią się straty mocy, które z kolei będą nasz przewodnik nagrzewać. Znając materiał i mase naszego przewodnika oraz czas takiego nagrzewania możemy:

Określić energię jaką dostarczymy, do objętości materiału, która przy stałej wartości strat mocy: $$ \Delta P = R\cdot I^{2} [W] $$

Będzie wynosić: $$ Q = \Delta P \cdot \tau [J] $$

a to z kolei pozwoli nam obliczyć przyrost temperatury naszego przewodnika przekształcając równanie definicyjne ciepła właściwego do postaci: $$ \Delta T = \frac{Q}{m \cdot c_{p}} [K] $$

W powyższym rozumowaniu kryje się jednak pewna pułapka, która jeżeli ją przeoczymy to nasze wyniki będą po prostu niepoprawne. Tym detalem o którym nie możemy zapominać jest fakt, że rezystancja przewodnika jest zależna od temperatury. $$ R(t) = R_{20}(1+\alpha(t-20^{o}C)) [\Omega] $$

Co jeżeli zbierzemy to razem do jednego sensownego równania: $$ t(\tau)-t_0 = \frac{R_{20}(1+\alpha(t(\tau)-20^{o}C))\cdot I^2 \cdot \tau}{m \cdot c_p} $$

Jeżeli dla przejrzystości naszego przykładu założyć $t_0 = 20^{o}C$

$$ \Delta t(\tau)= \frac{R_{20}(1+\alpha \Delta t(\tau))\cdot I^2 \cdot \tau}{m \cdot c_p} $$

A to nieco się komplikuje z perspektywy szybkiego rozwiązania, gdyż nasz postulat o stałości strat mocy w czasie nie jest już prawdziwy. W związku z tym zapisując to równanie dla elementarnego czasu $d\tau$ w którym taką stałość wspomnianych strat mocy możemy postulować, czyli przechodząc na wersję różniczkową można zapisać to jako:

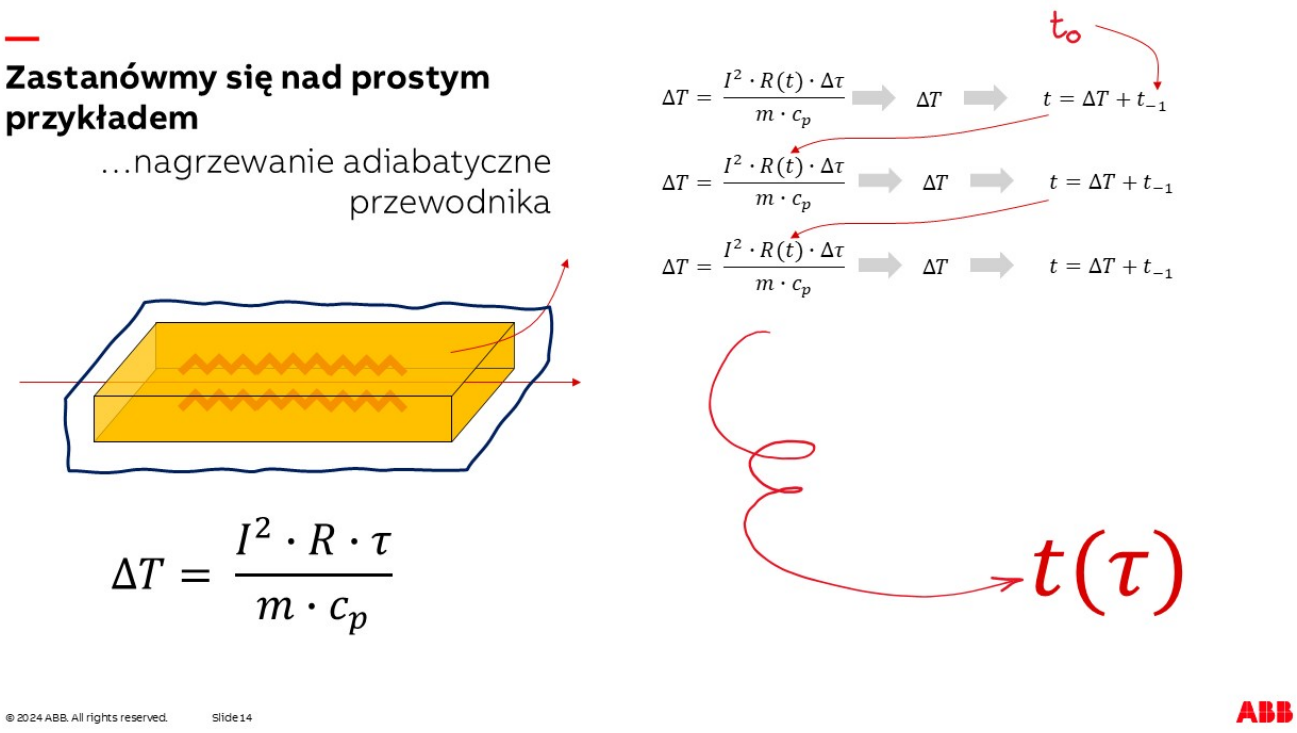
$$ \frac{d\Delta t(\tau)}{d\tau} = \frac{R_{20}(1+\alpha \Delta t(\tau))\cdot I^2 }{m \cdot c_p} $$

A po przejściu w domenę operatorową a później odwrotną transformacją Laplacea do domeny czasowej uzyskać finalnie wzór:

$$ \Delta t(\tau) = \frac{e^{\frac{R_{20}I^2 \cdot \alpha}{m \cdot c+p}\tau}-1}{\alpha} $$

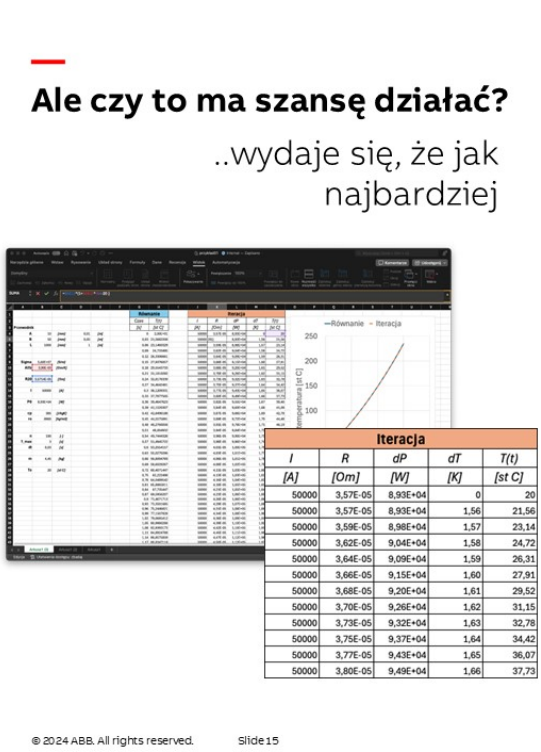
I ten jakże miły dla oka wzór, który uzyskać można w tak elegancki matematycznie sposób, stanowi niejako nasze rozwiązanie. A przy okazji jest on też przykładem wspominanego wcześniej **pierwszego** przybliżenia, czy kroku w naszym stopniowanym podejściu do analiz.

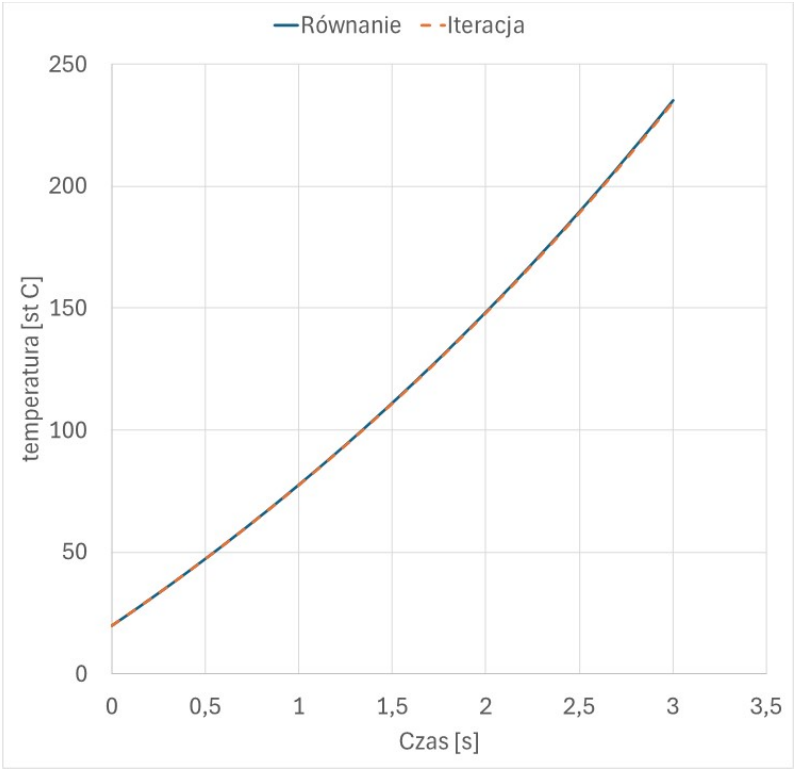
Jenak, skoro to nasze rozważanie ze względu na swój charakter musiało przybrać formę różniczkową, to można by też do niego podejść iteracyjnie. Czyli nie tyle różniczkowo co różnicowo.



Rozwiązując równanie w pierwotnej formie wielokrotnie, w każdym powtórzeniu przeliczając wartości rezystancji i strat mocy na podstawie temperatury obliczonej w poprzednim kroku.

Możemy to zrealizować na przykład w arkuszu kalkulacyjnym, a przy okazji porównać wyniki uzyskiwane za pomocą takiej krokowej metody przybliżonej do tych pochodzących z naszego, wyprowadzonego matematycznie wzoru.





Jak można zobaczyć, otrzymywane wyniki są bardzo zbliżone, acz nie dokładnie takie same. Dlaczego zatem użycie podejścia krokowego *iteracyjnego* może mieć jakieś zalety?

Na przykład dlatego, że nasze matematyczne rozwiązanie gdyby chcieć uwzględnić zależność innych wartości od czasu (*na przykład wartości prądu*) szybko może stać się bardzo skomplikowane, a na dodatek wymagać będzie kolejnego wyprowadzania równań.

Zaś nasz model *iteracyjny* niejako ze swojej natury jest pewnym rodzajem symulacji. I pozwala na zmianę parametrów w kolejnych jej krokach.

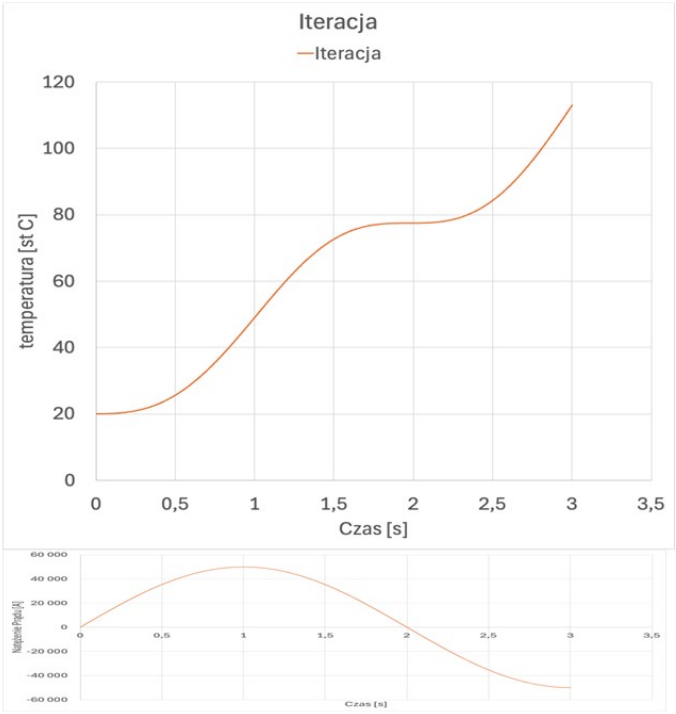
Na przykład postulując zmienną wartość wspomnianego natężenia prądu, możemy bez praktycznie żadnych modyfikacji naszego arkusza uzyskać odpowiedź termiczną analizowanego przewodnika.

Ale czy to ma szansę działać?

..to nawet ma pewne zalety

- Przejście do symulacji w domenie czasu pozwala na przykład porzucić założenie niezmienności parametrów analizowanego systemu
- Możemy na przykład przyjąć zmienną wartość prądu w naszym przewodniku…
- …dodać mechanizmy oddawania ciepła do otoczenia…

…i tak dalej.



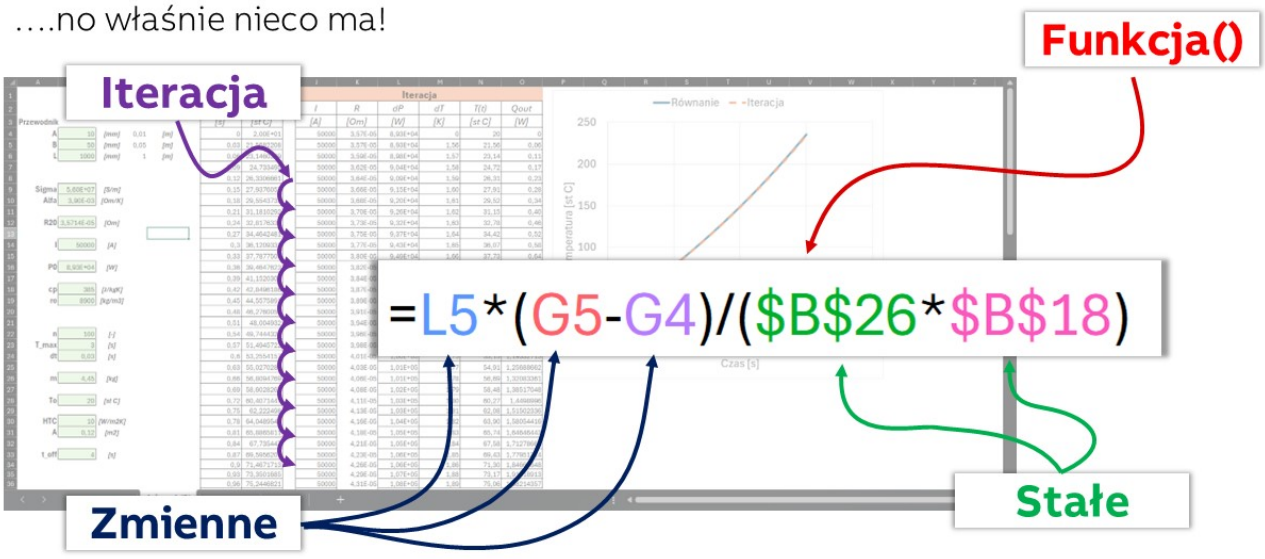
Ale gdzie tu jest programowanie?

Właśnie, na pierwszy rzut oka niewiele tutaj *zaprogramowaliśmy*. Przynajmniej w takim potocznym rozumieniu tego słowa. Ale czy aby na pewno?!?

Na tą implementację w arkuszu kalkulacyjnym można też popatrzeć nieco inaczej:

Tyko co to ma wspólnego z programowaniem?

….no właśnie nieco ma!



I dostrzegamy wtedy, że mamy tutaj zaskakujaco wiele podstawowych idei i mechanizmów znanych własnie z programowania:

- Funkcje

- Argumenty
- Stałe
- Zmienne
- Pętle iteracyjne

Innymi słowy **Właściwie to już napisaliśmy program!**

Python - żeby to programowanie było nieco bardziej *na serio*

Gdyby jednak chcieć porzucić pewne ograniczenia jakie narzuca arkusz kalkulacyjny jako środowisko pracy, albo po prostu zaprogramować nasze rozwiązanie nieco bardziej *jak programista*. To bardzo dobrym pierwszym wyborem będzie użycie języka programowania jakim jest **Python**.

Ale jeżeli ktoś chce, to można nieco bardziej

…czyli warto wiedzieć co jest dostępne

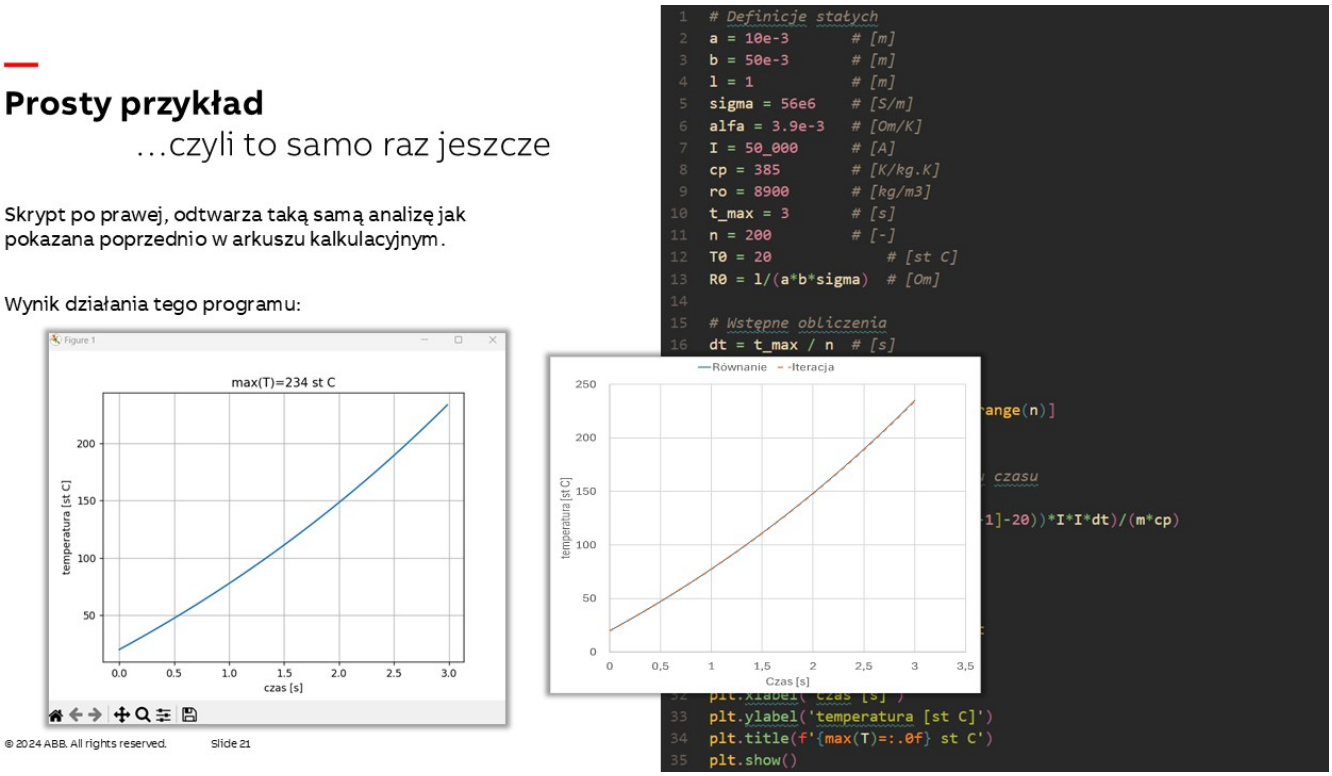
- **Python** w wersji 3 (aktualnie 3.13) jest interpretowanym językiem wysokiego poziomu.
- Jest to **darmowe** oprogramowanie **otwarto-źródłowe** oparte na licencji **PSF** License Agreement, która jest kompatybilna z licencją **GNU**.
- Języka Python **można** używać w projektach **komercyjnych**
- Ekosystem tego języka jest **bardzo bogaty** w **dodatkowe biblioteki** bezmiernie ułatwiające pracę i programowanie.

Implementując taką zamą metodę jak ta powyżej, ale już właśnie w pythonie:

```
# Definicje stałych
a = 10e-3           # [m]
b = 50e-3           # [m]
l = 1               # [m]
sigma = 56e6        # [S/m]
alfa = 3.9e-3       # [Om/K]
I = 50_000          # [A]
cp = 385            # [K/kg.K]
ro = 8900           # [kg/m3]
t_max = 3           # [s]
n = 200             # [-]
T0 = 20             # [st C]
R0 = l/(a*b*sigma)  # [Om]

# Wstępne obliczenia
dt = t_max / n      # [s]
m = a*b*l*ro        # [kg]


wektor_czasu = [i*dt for i in range(n)]
# wektor temperatury
T = [T0]


# obliczenia dla każdego elemtu wektora czasu
for tx in wektor_czasu[1:]:
    Tx = T[-1]+(R0*(1+alfa*(T[-1]-20))*I*I*dt)/(m*cp)
    T.append(Tx)
```

```
print(f'{max(T)=}')

# generowanie przebiegu
import matplotlib.pyplot as plt
plt.plot(wektor_czasu,T)
plt.grid()
plt.xlabel('czas [s]')
plt.ylabel('temperatura [st C]')
plt.title(f'{max(T)=:.0f} [st C]')
plt.show()
```

W wyniku działania którego dostajemy wyniki tożsame ilościowo z tymi z arkusza kalkulacyjnego.



Dzięki temu, że teraz mamy nasz program czy *skrypt* właśnie w takiej formie, możemy bardzo łatwo zrealizować kilka ciekawych modyfikacji.

Na start można by zapytać, czy ilość przyjętych kroków rozwiązania, a co za tym idzie wielkość pojedynczego kroku czasowego $d\tau$ oznaczonego w kodzie jako `dt` ma wpływ na wartość uzyskiwanego wyniku. Intuicja podpowiada, że pewnie ma. Jaki to jednak jest wpływ?

Możemy to sobie łatwo zobrazować, dokonując prostej modyfikacji naszego programu obejmując pętlę iteracyjną obliczeń

```
# obliczenia dla każdego elemtu wektora czasu
for tx in wektor_czasu[1:]:
    Tx = T[-1]+(R0*(1+alfa*(T[-1]-20))*I*I*dt)/(m*cp)
    T.append(Tx)
```

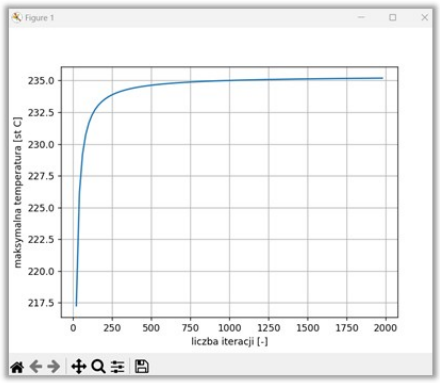
Jeszcze jedną pętlą, która będzie zmieniać ilość wykonywanych kroków iteracji - czyli w naszym kodzie długość wektora czasu `wektor_czasu`.

zmodyfikowany kod jest dostępny w pliku `przyklad01.py`

Prosty przykład

…więc go nieco skomplikujmy

- Jak sprawdzić czy wraz ze wzrostem liczby iteracji (n) uzyskana wartość maksymalna temperatury się zmienia?
- Czy ta zmienność jest zbieżna do wartości ustalonej?



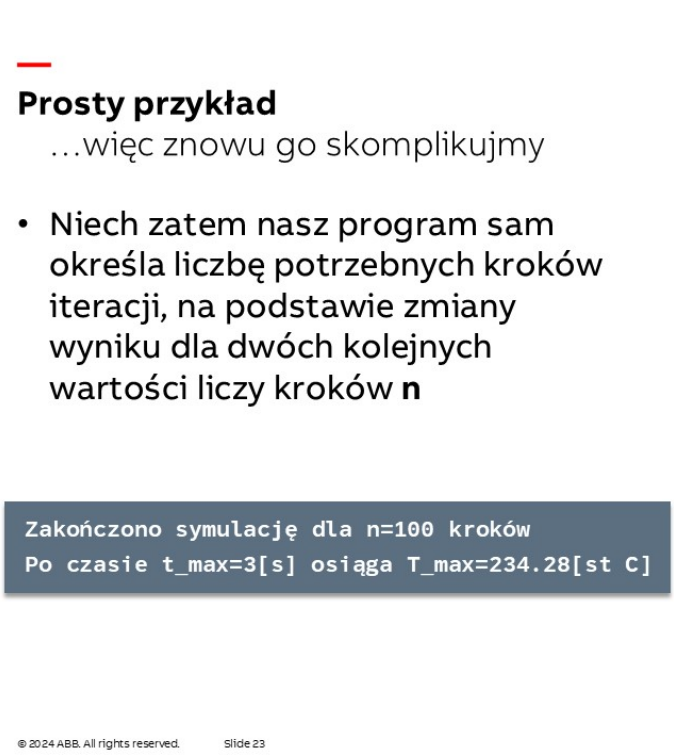
 Slide 22

```python
# Wstępne obliczenia
dt = t_max / n  # [s]
m = a*b*l*ro    # [kg]

T_max = []
N = [n for n in range(20,2000,20)]

for n in N:
    T = [T0]
    dt = t_max / n
    wektor_czasu = [i*dt for i in range(n)]

    for tx in wektor_czasu[1:]:
        Tx = T[-1]+(R0*(1+alfa*(T[-1]-20))*I*I*dt)/(m*cp)
        T.append(Tx)

    T_max.append(max(T))

# generowanie wykresu
import matplotlib.pyplot as plt
plt.plot(N,T_max)
plt.grid()
plt.xlabel('liczba iteracji [-]')
plt.ylabel('maksymalna temperatura [st C]')
plt.show()
```

Wykonanie tego skryptu, pokazuje nam, że wartość wyniku zalezy od ilości kroków iteracji, jednak wraz z jej wzrostem staje się **zbieżna** do pewnej wartości, albo innymi słowy **stabilizuje się**. To z kolei sugeruje, że nie ma sensu przesadzać z ilością kroków iteracji, gdyz wydłuża to tylko działanie naszego kodu, a nie przynosi już szczególnych korzyści.

Można się nawet pokusić o zautomatyzowanie dobierania potrzebnej ilości kroków, na przykład zwiększania tejże aż do momentu, gdy dalsze zwiększanie powoduje zmianę wartości wyniku obliczeń mniejszą niż założona `delta`.

Implementację takiego mechanizmu umieściłem w pliku `przyklad03.py`.




```python
# Definicje statych
a = 10e-3           # [m]
b = 50e-3           # [m]
l = 1               # [m]
sigma = 56e6        # [S/m]
alfa = 3.9e-3       # [Om/K]
I = 50_000          # [A]
cp = 385            # [K/kg.K]
ro = 8900           # [kg/m3]
t_max = 3           # [s]
n = 200             # [-]
T0 = 20             # [st C]
R0 = 1/(a*b*sigma)  # [Om]

# Wstepne obliczenia
m = a*b*l*ro        # [kg]


delta = 1/1000 # [-] maksymalna dopuszczalna zmiana wyniku
T_max = 1      # [st C] wartość startowa

N = [n for n in range(20,1000,20)]
for n in N:
    T_prev = T0
    dt = t_max / n

    for tx in range(n):
        T_prev = T_prev+(R0*(1+alfa*(T_prev-20))*I*I*dt)/(m*cp)

    d = abs((T_prev - T_max)/T_prev)

    if d < delta:
        print(f'Zakończono symulację dla {n=} kroków')
        break

    T_max = T_prev

print(f'Po czasie {t_max=}[s] osiąga {T_max=:.2f}[st C]')
```

I tak właśnie zaczynając od równania i arkusza kalkulacyjnego, zbudowaliśmy już całkiem ładnie zachowujacy się **program**, który stanowi już prawie gotową małą aplikację.

To co można by jeszcze zmienić, to nieco usystematyzować nasz kod, zapisać go w sposób nieco bardziej zgodny z przyjetymi w *pythonie* zasadami pisania w miarę przejrzystego kodu. A na dodatek, pozwolić użytkownikowi na interakcję z programem bez konieczności zmiany samego pliku.

To ostatnie możemy zrealizować za pomoca pobierania parametrów analizy z linii komend, zamiast wpisywania ich do naszego skryptu jako stałe. Tutaj z pomoca przychodzi gotowa i domyślnie dostępna w pythonie biblioteka `argprase`. Całą implementację można znaleźć w pliku `aplikacja.py` a jej działanie jest następujące:

Prosty przykład

…i to już ostatnia komplikacja

Zmieńmy nasz dotychczasowy skrypt już w prawie aplikację:

- Pobierajmy stałe jako parametry z linii komend
- Nieco uprzątnijmy nasz kod.

```
>> python .\aplikacja.py

Zakończono symulację dla n=100 kroków
Po czasie t_max=3[s] osiąga T_max=234.28[st C]
```

```
>> python .\aplikacja.py --sigma 32e6

Zakończono symulację dla n=140 kroków
Po czasie t_max=3[s] osiąga T_max=505.41[st C]
```

```
>> python .\aplikacja.py --a 20e-3 --b 50e-3

Zakończono symulację dla n=60 kroków
Po czasie t_max=3[s] osiąga T_max=62.14[st C]
```

 Slide 24

```python
import argparse

def pobierz_parametry():
    parser = argparse.ArgumentParser(description="Pobieranie stałych z linii komend")
    parser.add_argument('--a', type=float, default=10e-3, help='Wartość a [m]')
    parser.add_argument('--b', type=float, default=50e-3, help='Wartość b [m]')
    parser.add_argument('--l', type=float, default=1, help='Wartość l [m]')
    parser.add_argument('--sigma', type=float, default=56e6, help='Wartość sigma [S/m]')
    parser.add_argument('--alfa', type=float, default=3.9e-3, help='Wartość alfa [Om/K]')
    parser.add_argument('--I', type=float, default=50_000, help='Wartość I [A]')
    parser.add_argument('--cp', type=float, default=385, help='Wartość cp [K/kg.K]')
    parser.add_argument('--ro', type=float, default=8900, help='Wartość ro [kg/m3]')
    parser.add_argument('--t_max', type=float, default=3, help='Wartość t_max [s]')
    parser.add_argument('--n', type=int, default=200, help='Wartość n [-]')
    parser.add_argument('--T0', type=float, default=20, help='Wartość T0 [st C]')
    parser.add_argument('--delta', type=float, default=1/1000, help='Maksymalna dopuszczalna zmiana wyniku [-]')

    return parser.parse_args()

def main():

    args = pobierz_parametry()

    R0 = args.l/(args.a*args.b*args.sigma)  # [Om]
    m = args.a*args.b*args.l*args.ro        # [kg]
    mcp = m*args.cp

    T0 = args.T0
    t_max = args.t_max
    alfa = args.alfa
    I = args.I
    delta = args.delta

    T_max = 1      # [st C] wartość startowa
    N = [n for n in range(20,10000,20)]
    for n in N:
        T_prev = T0
        dt = t_max / n

        for tx in range(n):
            T_prev = T_prev+(R0*(1+alfa*(T_prev-20))*I*I*dt)/(mcp)

        d = abs((T_prev - T_max)/T_prev)

        if d < delta:
            print(f'Zakończono symulację dla {n=} kroków')
            break

        T_max = T_prev

    print(f'Po czasie {t_max=}[s] osiąga {T_max=:.2f}[st C]')

if __name__ == "__main__":
    main()
```

I tym sposobem stworzyliśmy nasza aplikację!

Jeszcze kilka słów końcowych

Przykłady aplikacji jakie miałem okazję stworzyć w tym środowisku na potrzeby moich zadań inzynierskich

CDS

Cross Section Designer

Quasi-FEM program pozwalający na analizę rozkładu gęstości prądu w przekroju układu 1/2/3 fazowego przewodników.

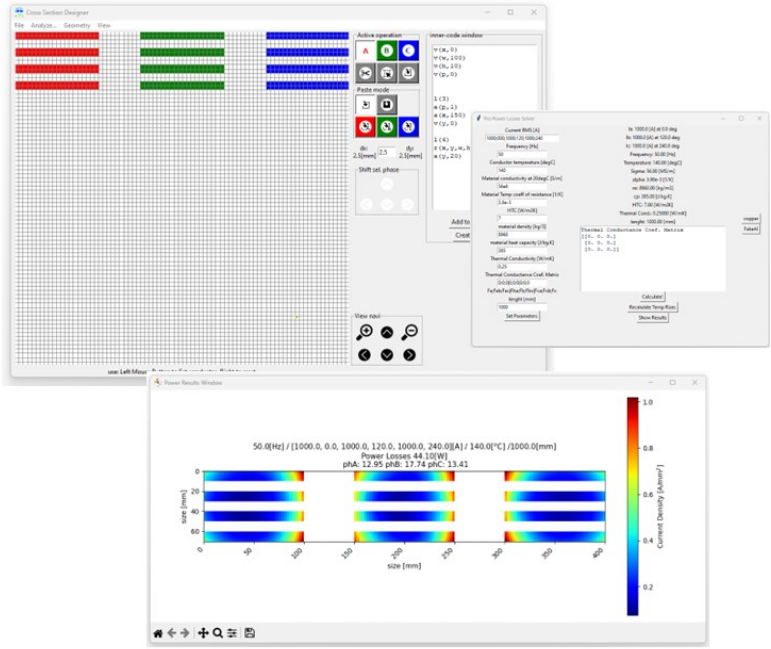
python™

tkinter

NumPy

matplotlib

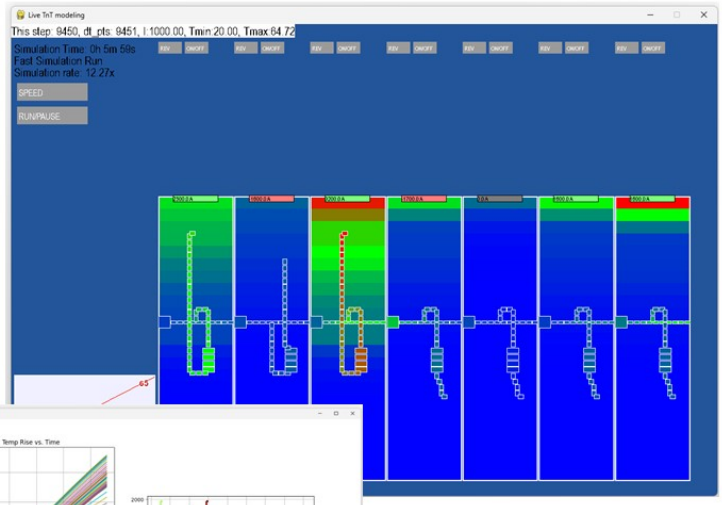
Numba

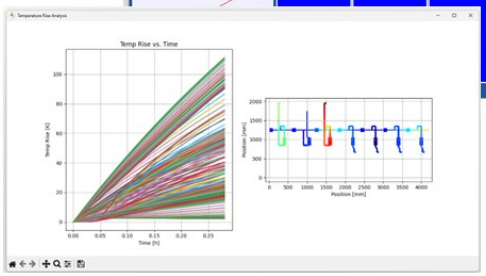


 Slide 28

TnT2.0
Thermal Network Tool 2.0

Iteracyjny system solwera Sieci Termicznych pozwalający na symulowanie termicznego zachowania się rozdzielnic i aparatów elektrycznych

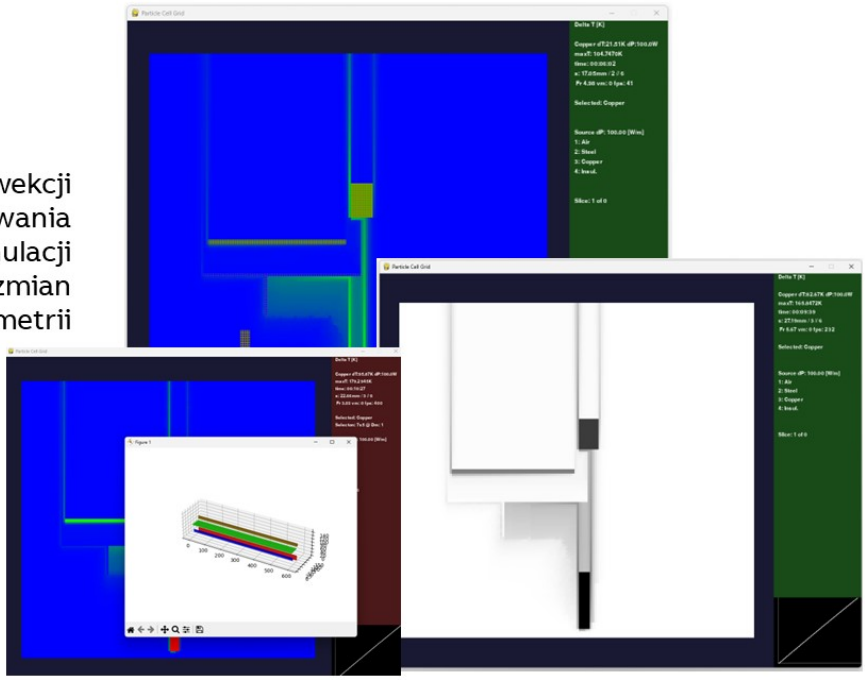




 Slide 29

PGC
Particle Grid Cell

Sudo-symulator naturalnej konwekcji stworzony celem opracowania wydajnego modelu symulacji chłodzenia w zależności od zmian geometrii

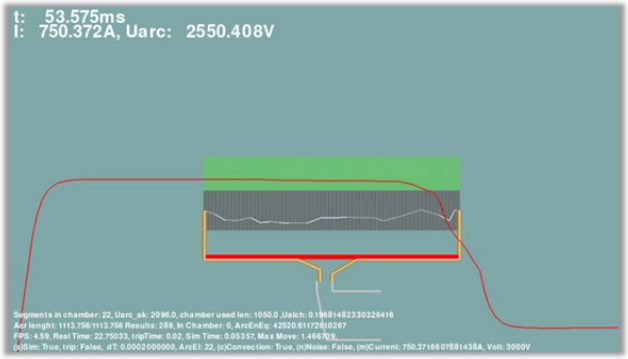


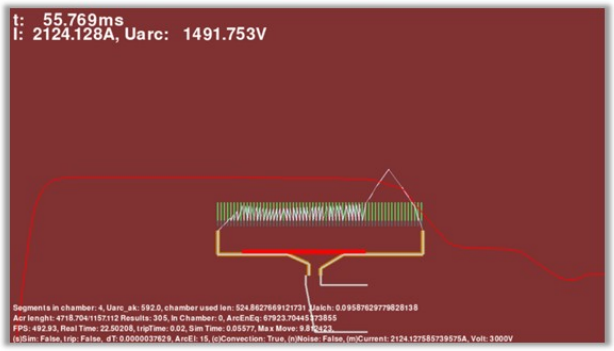
 Slide 30

ElektroGumka

…cóż nazwa sama się tłumaczy

Sudo-symulator zachowania się łuku elektrycznego w komorze gaszeniowej wyłącznika prądu stałego





python™ NumPy matplotlib Numba pygame

 Slide 31

ABB

Kilka słów ostrzeżenia przed skutkami ubocznymi

Możemy polubić pracę w terminalu

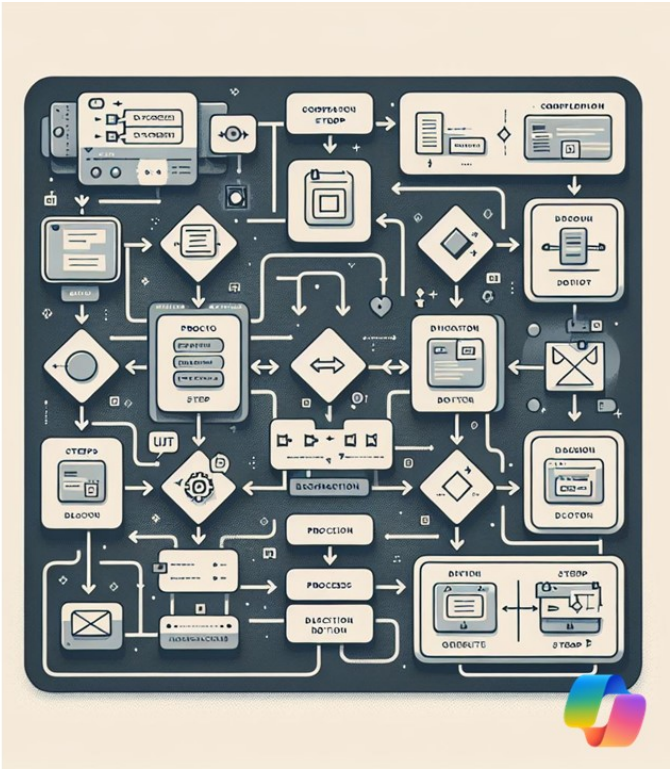
…i być przez to postrzegani jak *haker* lub informatyczny *czarodziej*.

Ludzie mogą zacząć od nas oczekiwać, że potrafimy rozwiązać każdy problem związany z komputerami

 Slide 33

Programowanie uczy uporządkowanego myślenia i takiego właśnie podejścia do rozwiązywania problemów

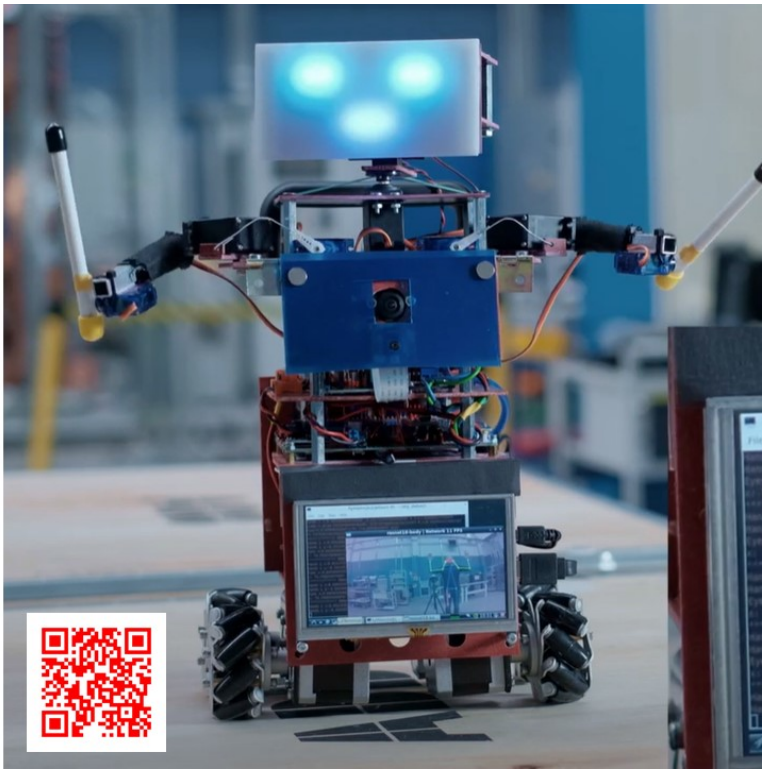
Wiele zasad związanych z pisaniem przyzwoitego jakościowo kodu można ekstrapolować na inne dziedziny inżynierii



 Slide 34

Może okazać się, że jest to bardzo wciągające zajęcie, które ma potencjał do stania się swoistą króliczą norą…

Dla mnie skończyło się programowaniem robotów z wykorzystaniem modeli ML i ComputerVision…



 Slide 35

Podsumowując

Pisanie programu, skryptu czy kodu na własne potrzeby nie jest wcale trudne

Z perspektywy technologicznej zasadniczo nie ma bariery wejścia

Może to być nie tylko przydatne ale i bardzo ciekawe

Dla nie informatyków – jest to pouczające doświadczenie, które uważam jest bardzo wartościowe

Dziękuję, Tomasz