Wstęp

Ta prezentacja.

- PDF
- HTML (jasny)
- reveal (jasny)
- reveal (ciemny)
- deck.js (jasny)

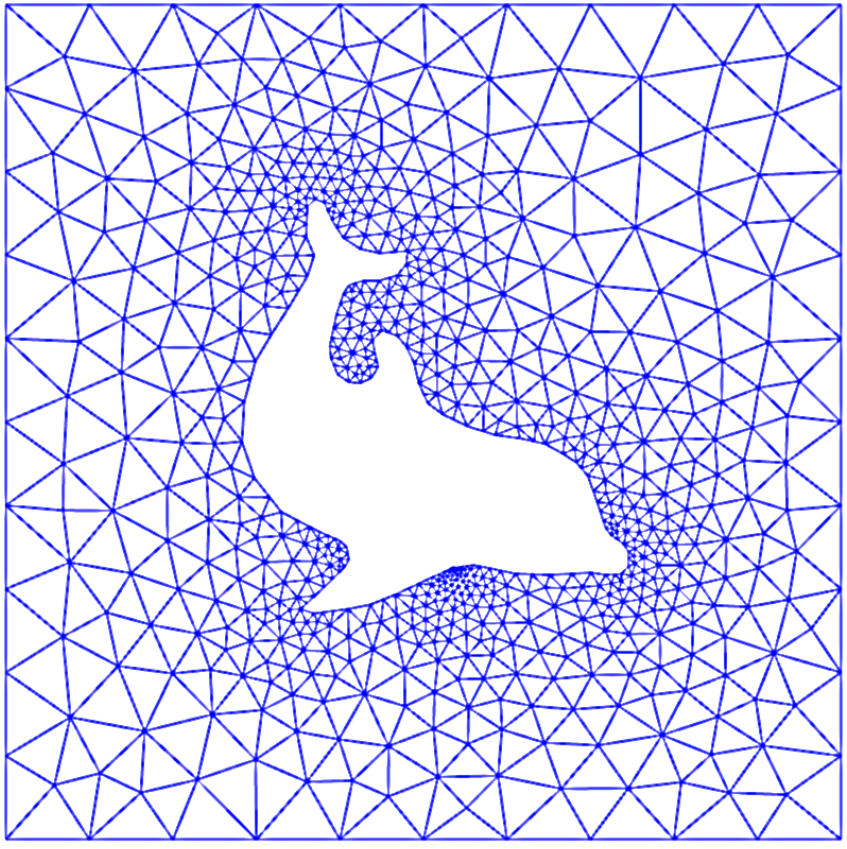
Kody Pythona. Repozytorium

Hans Petter Langtangen (1962-2016).

- Strona domowa
- Github
- DocOnce i książki HPL
- Książka o FEM (EN)

Metoda elementów skończonych - wstęp

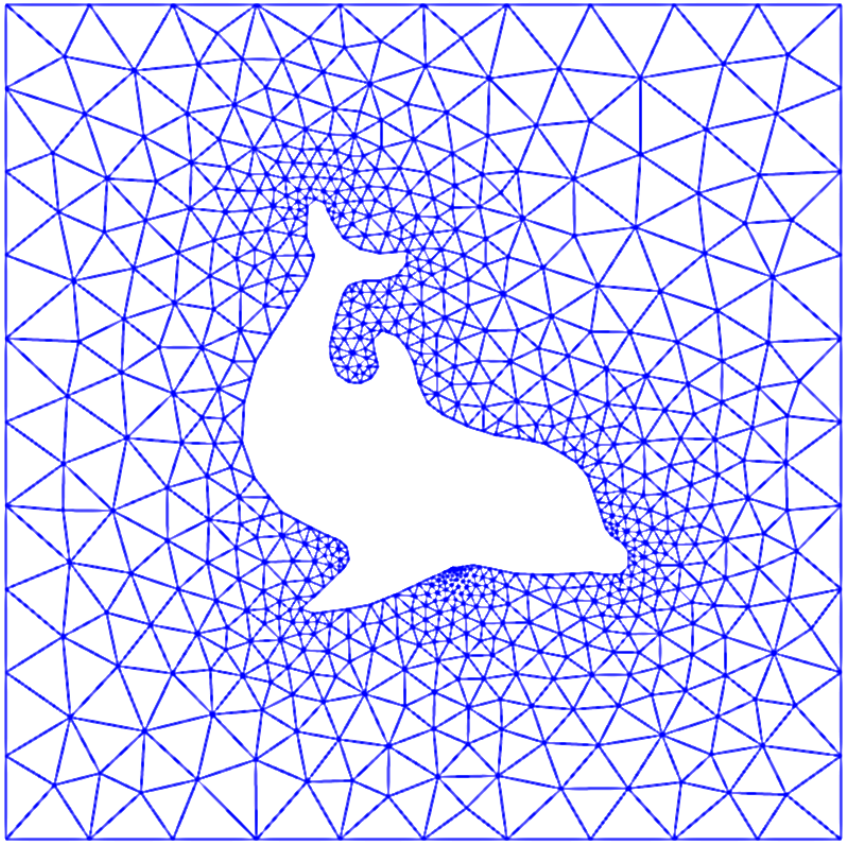
Finite Element Method, FEM, MES



- pozwala rozwiązywać RRCz dla obszarów o złożonej geometrii
- pozwala "regulować" dokładność siatki tam gdzie to potrzebne
- możliwość użycia aproksymacji wyższego rzędu
- dobre, matematyczne podstawy - co pozwala na dokładną analizę metody

Metoda elementów skończonych - wstęp

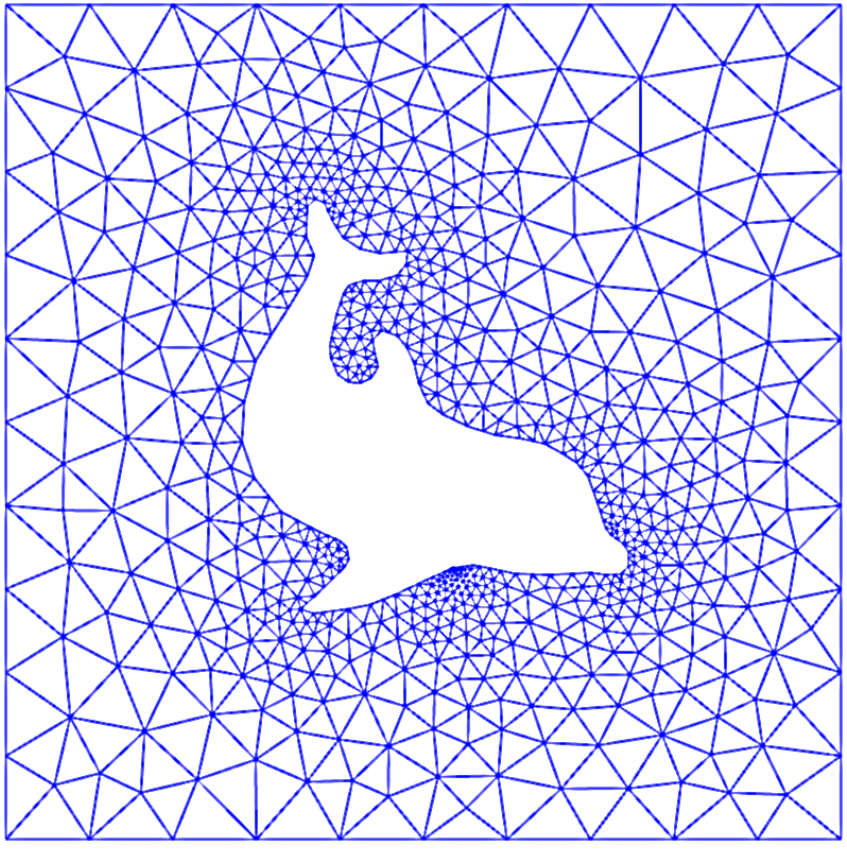
Finite Element Method, FEM, MES



- pozwala rozwiązywać RRCz dla obszarów o złożonej geometrii
- pozwala "regulować" dokładność siatki tam gdzie to potrzebne
- możliwość użycia aproksymacji wyższego rzędu
- dobre, matematyczne podstawy - co pozwala na dokładną analizę metody

Metoda elementów skończonych - wstęp

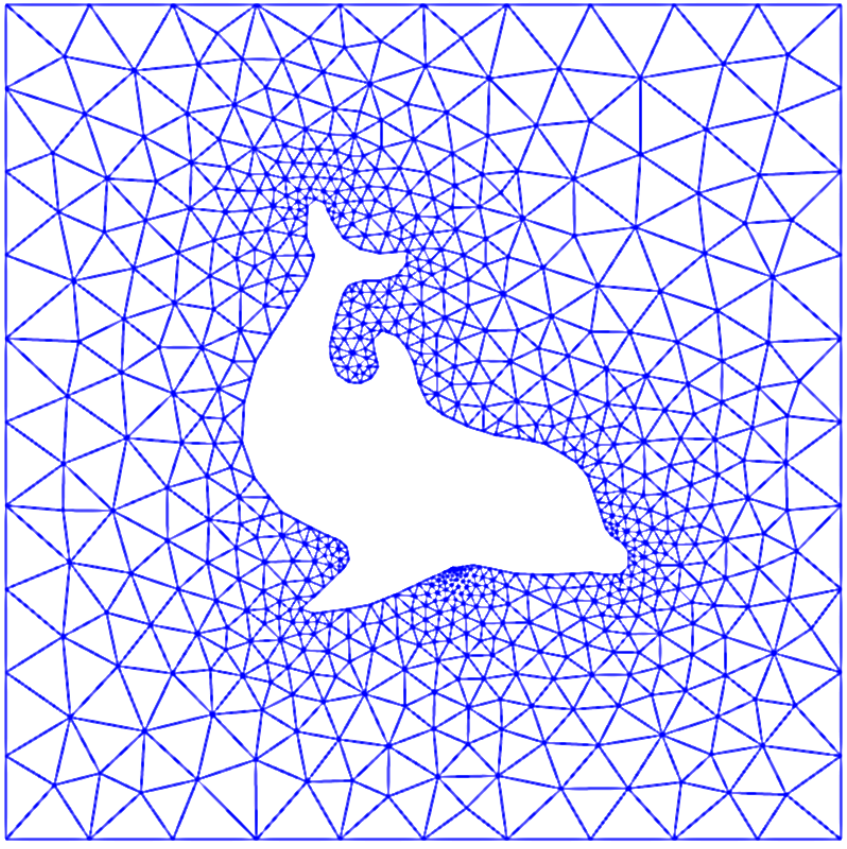
Finite Element Method, FEM, MES



- pozwala rozwiązywać RRCz dla obszarów o złożonej geometrii
- pozwala "regulować" dokładność siatki tam gdzie to potrzebne
- możliwość użycia aproksymacji wyższego rzędu
- dobre, matematyczne podstawy - co pozwala na dokładną analizę metody

Metoda elementów skończonych - wstęp

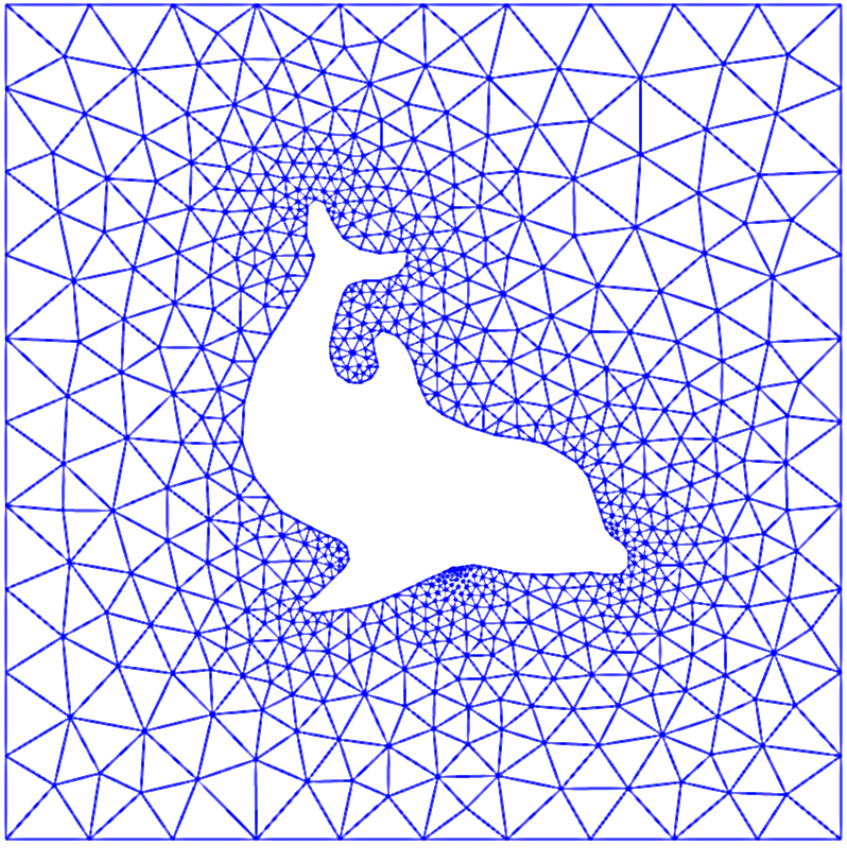
Finite Element Method, FEM, MES



- pozwala rozwiązywać RRCz dla obszarów o złożonej geometrii
- pozwala "regulować" dokładność siatki tam gdzie to potrzebne
- możliwość użycia aproksymacji wyższego rzędu
- dobre, matematyczne podstawy - co pozwala na dokładną analizę metody

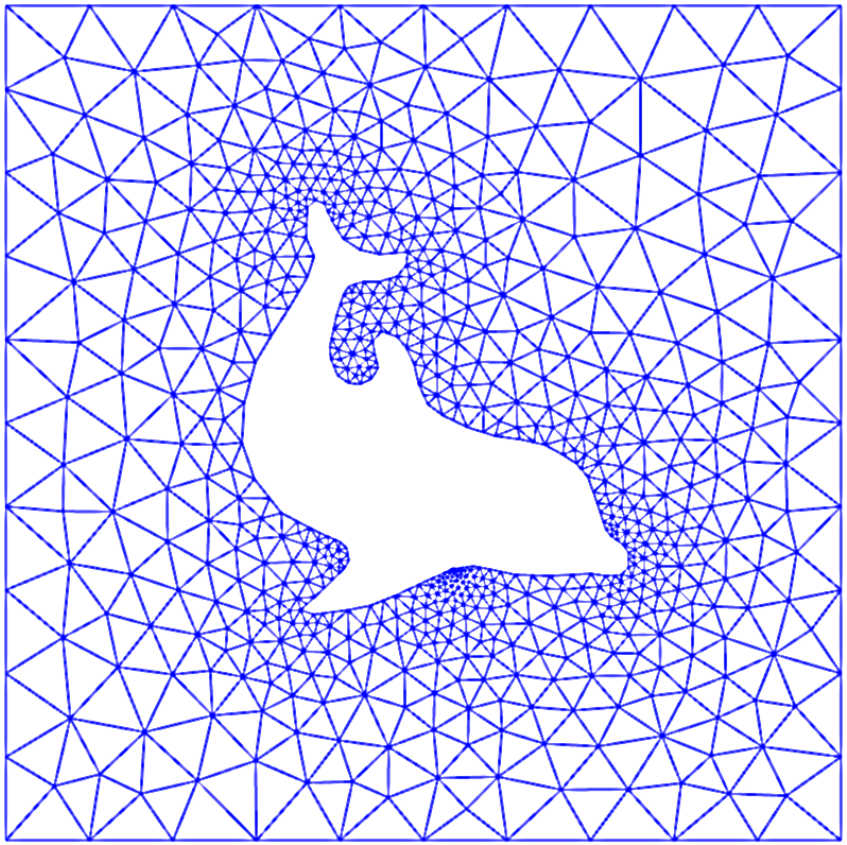
Metoda elementów skończonych - wstęp

Finite Element Method, FEM, MES



- pozwala rozwiązywać RRCz dla obszarów o złożonej geometrii
- pozwala "regulować" dokładność siatki tam gdzie to potrzebne
- możliwość użycia aproksymacji wyższego rzędu
- dobre, matematyczne podstawy - co pozwala na dokładną analizę metody

Delfin



Rozwiązywanie RRCz przy pomocy FEM

Zagadnienia stacjonarne:

1. Przekształcenie zagadnienia brzegowego do *postaci wariacyjnej*
2. Zdefiniowanie funkcji aproksymujących dla *elementów skończonych*
3. Przkształcenie zagadnienia ciągłego w dyskretne wyrażone *układem równań liniowych* (URL)
4. Rozwiązanie URL

Zagadnienia niestacjonarne (zależne od czasu):

- MES - aproksymacja przestrzeni
- FDM (lub metoda rozw. ODE) - aproksymacja w czasie

Etapy nauki FEM

Jak?

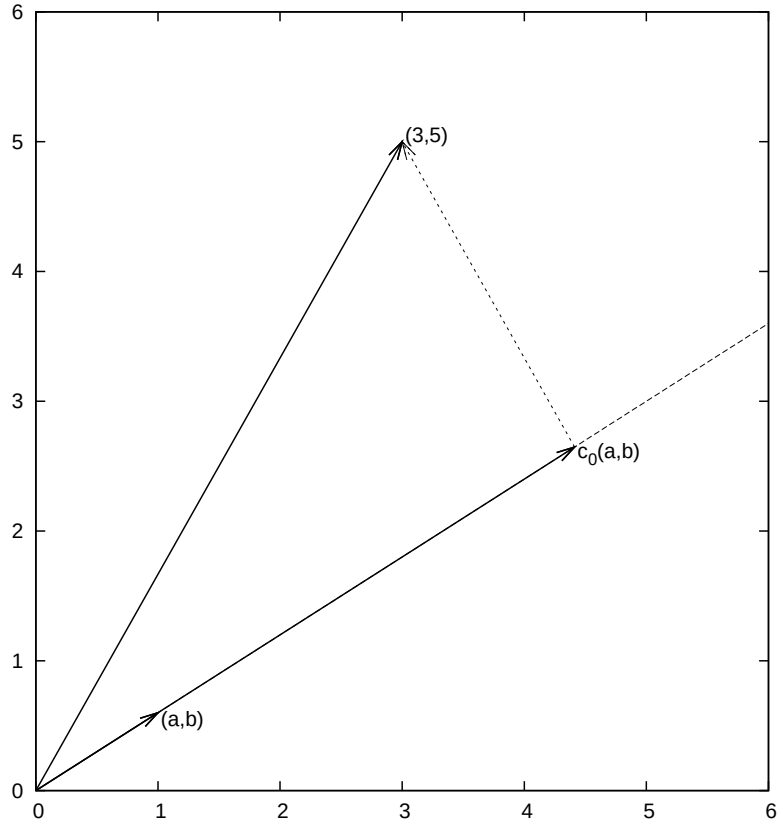
1. Elementy teoria aproksymacji, a nie rozw. RRCz
2. Wstęp do aproksymacji elementami skończonymi
3. W końcu zastosowanie powyższego do rozw. RRCz

Dlaczego tak?

Istnieje wiele wariantów i odmian FEM. Dzięki proponowanemu podejściu łatwiej się "połapać". Unikamy zamieszania wielością podejść do tematu, koncepcji i technicznych/implementacyjnych szczegółów.

Aproksymacja w przestrzeniach wektorowych

Jak znaleźć wektor pewnej przestrzeni, który aproksymuje wektor przestrzeni o większym wymiarze?



Aproksymacja jako kombinacja liniowa założonych funkcji bazowych

Ogólna idea poszukiwania elementu (wektora/funkcji) $u(x)$ pewnej przestrzeni przybliżającego zadany element $f(x)$:

$$u(x) = \sum_{i=0}^{N} c_I \psi_i(x)$$

gdzie

- $\psi_i(x)$ założone funkcje
- c_i, $i = 0, \ldots, N$ nieznane współczynniki (do wyznaczenia)

Trzy główne sposoby wyznaczania niewiadomych współczynników

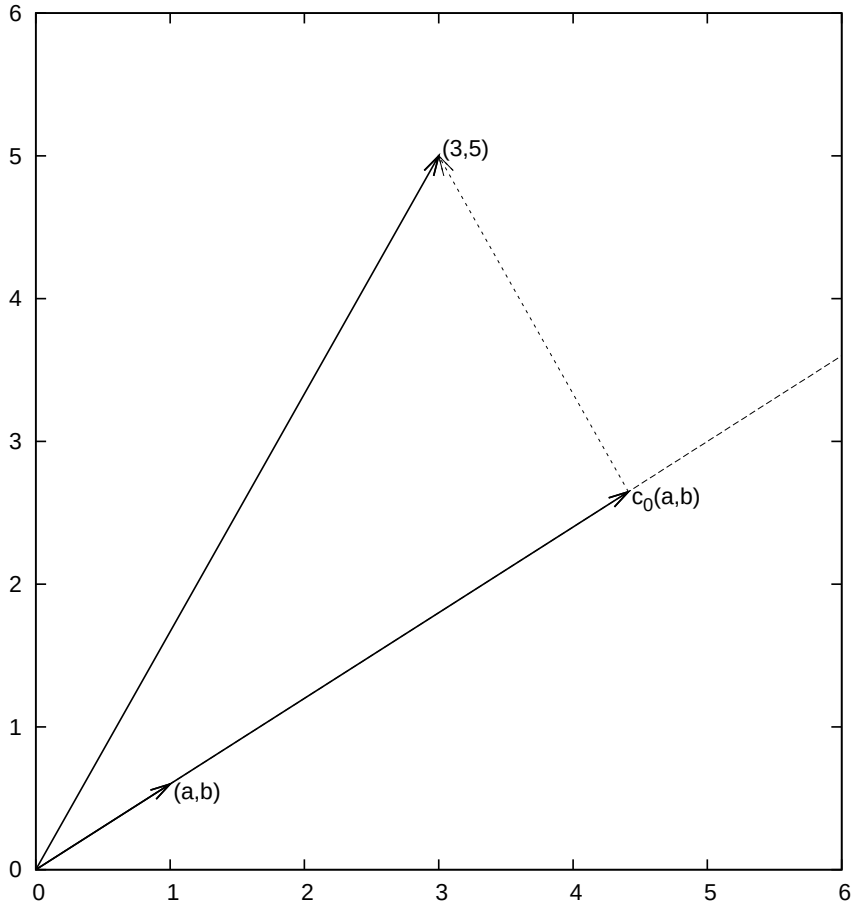
- metoda najmniejszych kwadratów (ang. Least Squares Method LSM)
- metoda Galerkina
- metoda kolokacji

Sposób opisu i notacja zaproponowane w materiałach wybrane w sposób ułatwiający zrozumienie open-source'owego pakietu FEniCS do obliczeń metodą elementów skończonych.

Aproksymacja wektorów: przykład – aproksymacja na płaszczyznie

Zadanie:

Znajdź przybliżenie wektora $\boldsymbol{f} = (3, 5)$ wzdłuż zadanego kierunku.



analogie: element - punkt - wektor - funkcja

Aproksymacja wektorów: przestrzenie wektorowe – terminologia

$$V = \text{span}\{\psi_0\}$$

- ψ_0 wektor bazowy przestrzeni V
- Znajdź $\boldsymbol{u} = c_0\psi_0 \in V$
- Jak wyznaczyć c_0 tak, aby $\boldsymbol{u}$ "najlepiej" przybliżało $\boldsymbol{f}$?
- Wizualnie rozwiązanie narzuca się samo
- Jak sformułować to ogólnie, w języku matematyki?

Niech

- $\boldsymbol{e} = \boldsymbol{f} - \boldsymbol{u}$ to błąd
- $(\boldsymbol{u}, \boldsymbol{v})$ – iloczyn skalarny wektorów
- $||\boldsymbol{e}|| = \sqrt{(\boldsymbol{e}, \boldsymbol{e})}$ – norma błędu (jaka? (normy p-te))

Uwaga:

(a, b) – punkt/wektor przestrzeni (dwuwymiarowej)
$(\boldsymbol{u}, \boldsymbol{v})$ – iloczyn skalarny dwóch wektorów przestrzeni (dowolnie-wymiarowej)

Metoda najmniejszych kwadratów - idea

- Idea: znaleźć c_0 takie, aby $||\boldsymbol{e}||$ był minimalizowany (jak najmniejszy/najkrótszy)
- Dla wygody (matematycznej): minimalizujemy $E = ||\boldsymbol{e}||^2$

$$\frac{\partial E}{\partial c_0} = 0$$

Metoda najmniejszych kwadratów - obliczenia

$$E(c_0) = (\boldsymbol{e}, \boldsymbol{e}) = (\boldsymbol{f} - \boldsymbol{u}, \boldsymbol{f} - \boldsymbol{u}) = (\boldsymbol{f} - c_0\psi_0, \boldsymbol{f} - c_0\psi_0)$$
$$= (\boldsymbol{f}, \boldsymbol{f}) - 2c_0(\boldsymbol{f}, \psi_0) + c_0^2(\psi_0, \psi_0)$$

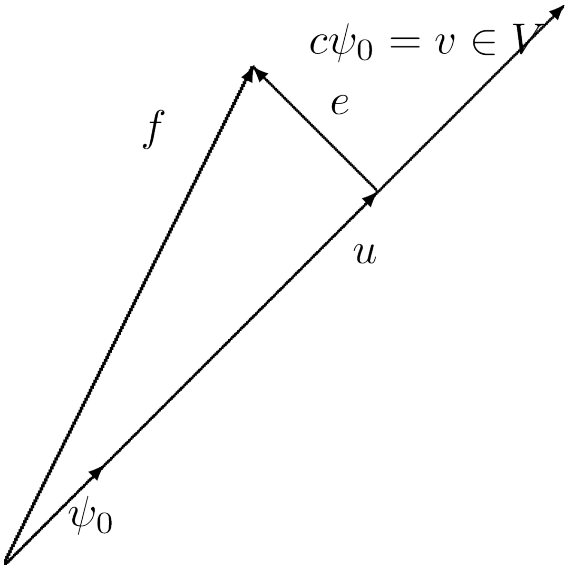
$$\frac{\partial E}{\partial c_0} = -2(\boldsymbol{f}, \psi_0) + 2c_0(\psi_0, \psi_0) = 0 \qquad (1)$$

$$c_0 = \frac{(\boldsymbol{f}, \psi_0)}{(\psi_0, \psi_0)} = \frac{3a + 5b}{a^2 + b^2}$$

Spostrzeżenie (na później): warunek znikania pochodnej (1) można równoważnie zapisać jako:

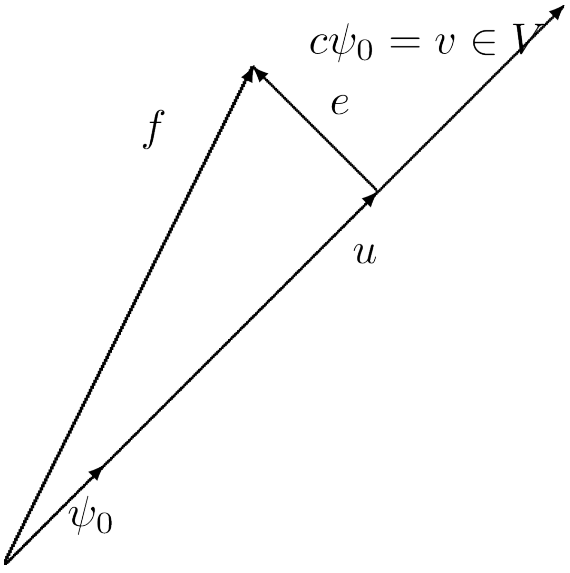
$$(\boldsymbol{e}, \psi_0) = 0$$
$$(\boldsymbol{e}, \psi_0) = (\boldsymbol{f} - \boldsymbol{u}, \psi_0) = (\boldsymbol{f}, \psi_0) - (\boldsymbol{u}, \psi_0) \ldots . \qquad (* - 2 \ldots)$$
$$[\boldsymbol{f} - \boldsymbol{u}] \cdot \begin{bmatrix} \psi_0 \end{bmatrix} = [\quad \boldsymbol{f} \quad] \cdot \begin{bmatrix} \psi_0 \end{bmatrix} - [\quad \boldsymbol{u} \quad] \cdot \begin{bmatrix} \psi_0 \end{bmatrix}$$

Metoda Galerkina



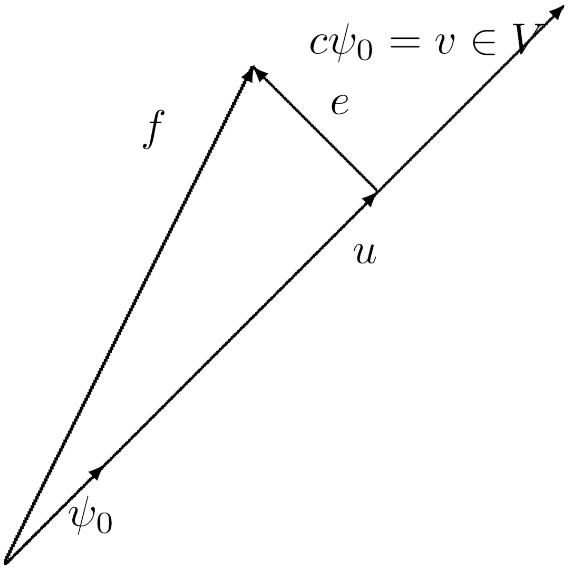
- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$ (rysunek)
- $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\boldsymbol{\psi_0}$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



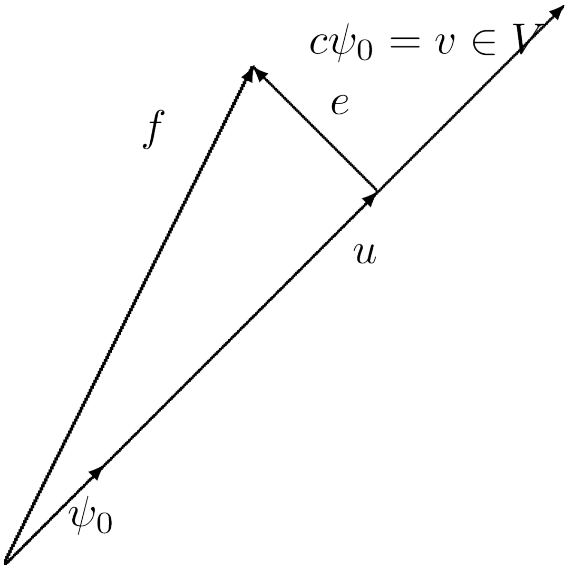
- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$ (rysunek)
- $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\boldsymbol{\psi_0}$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



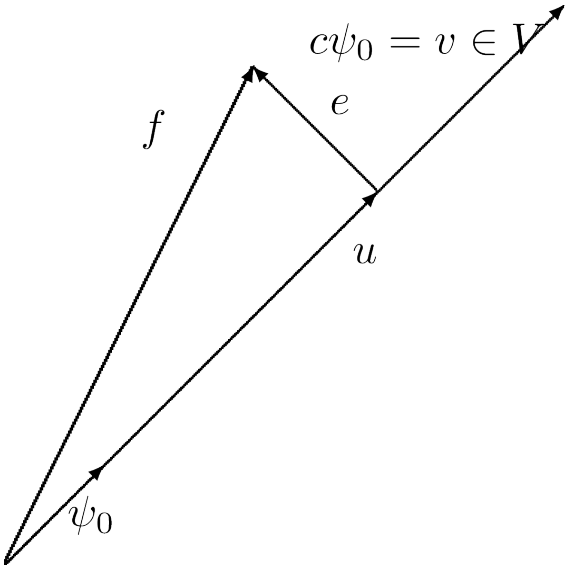
- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \psi_0) = 0$ (rysunek)
- $(\boldsymbol{e}, \psi_0) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \psi_0) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\psi_0$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



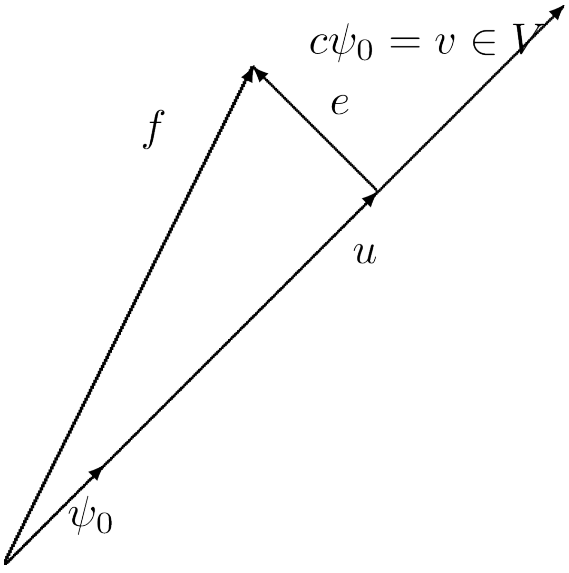
- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \psi_0) = 0$ (rysunek)
- $(\boldsymbol{e}, \psi_0) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \psi_0) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\psi_0$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



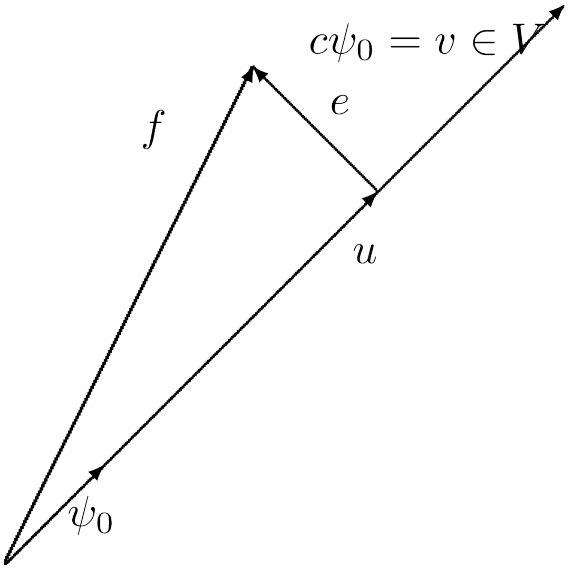
- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \psi_0) = 0$ (rysunek)
- $(\boldsymbol{e}, \psi_0) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \psi_0) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\psi_0$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



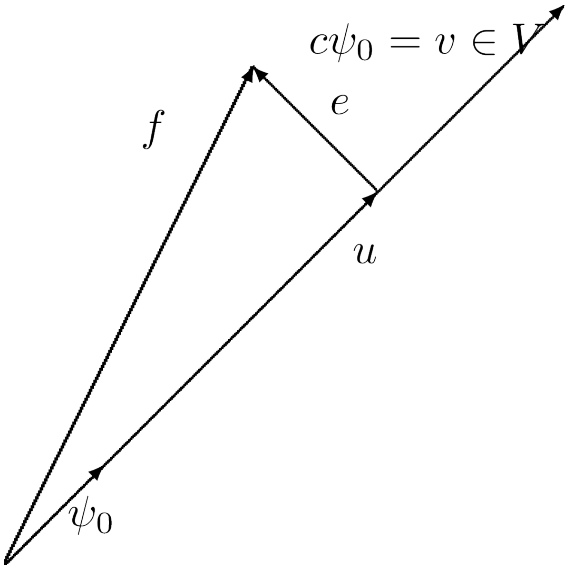
- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$ (rysunek)
- $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\boldsymbol{\psi_0}$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$ (rysunek)
- $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\boldsymbol{\psi_0}$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Metoda Galerkina



- $\min E \rightarrow$ to samo co: $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$ (rysunek)
- $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0 \Rightarrow (\boldsymbol{e}, \boldsymbol{v}) = 0$ dla *dowolnego* $\boldsymbol{v} \in V$ (rysunek)
- Czyli: zamiast korzystać z aproksymacji średniokwadratowej, można wymusić, aby $\boldsymbol{e}$ było ortogonalne (prostopadłe) dla dowolnego $\boldsymbol{v} \in V$ – oczywiste na rysunku a...
- W języku matematyki: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$
- Równoważnie: znajdź takie c_0 aby $(\boldsymbol{e}, \boldsymbol{\psi_0}) = 0$
- Podstawmy: $\boldsymbol{e} = \boldsymbol{f} - c_0\boldsymbol{\psi_0}$ i rozwiążmy ze względu na c_0
- Ostatecznie: to samo równanie (a więc i to samo rozwiązanie) co w metodzie najmniejszych kwadratów

Aproksymacja wektora przestrzeni dowolniewymiarowej

Dla danego wektora $\boldsymbol{f}$, znajdź przybliżenie $\boldsymbol{u} \in V$:

$$V = \text{span}\,\{\psi_0, \ldots, \psi_N\}$$

(span czyt. przestrzeń rozpięta na wektorach...)

Mając dany zbiór wektorów niezależnych liniowo $\psi_0, \ldots, \psi_N$, dowolny wektor $\boldsymbol{u} \in V$ można zapisać jako:

$$\boldsymbol{u} = \sum_{j=0}^{N} c_j \psi_j$$

Przykład

TODO

Metoda najmniejszych kwadratów

Idea: znaleźć takie $c_0, \ldots, c_N$, aby $E = ||\boldsymbol{e}||^2$ był minimalizowany, $\boldsymbol{e} = \boldsymbol{f} - \boldsymbol{u}$.

$$E(c_0, \ldots, c_N) = (\boldsymbol{e}, \boldsymbol{e}) = (\boldsymbol{f} - \sum_j c_j \psi_j, \boldsymbol{f} - \sum_j c_j \psi_j)$$

$$= (\boldsymbol{f}, \boldsymbol{f}) - 2\sum_{j=0}^{N} c_j(\boldsymbol{f}, \psi_j) + \sum_{p=0}^{N}\sum_{q=0}^{N} c_p c_q (\psi_p, \psi_q)$$

$$\frac{\partial E}{\partial c_i} = 0, \quad i = 0, \ldots, N$$

Po odrobinie obliczeń otrzymuje się *układ równań liniowych*:

$$\sum_{j=0}^{N} A_{i,j} c_j = b_i, \quad i = 0, \ldots, N \tag{2}$$

$$A_{i,j} = (\psi_i, \psi_j) \tag{3}$$

$$b_i = (\psi_i, \boldsymbol{f}) \tag{4}$$

Metoda Galerkina

Można pokazać, że poszukiwanie minimalnego $||\boldsymbol{e}||$ jest równoważne poszukiwaniu $\boldsymbol{e}$ ortogonalnego do wszystkich $\boldsymbol{v} \in V$:

$$(\boldsymbol{e}, \boldsymbol{v}) = 0, \quad \forall \boldsymbol{v} \in V$$

co jest równoważne temu aby $\boldsymbol{e}$ był ortogonalny do każdego wektora bazowego:

$$(\boldsymbol{e}, \psi_i) = 0, \quad i = 0, \ldots, N$$

Warunek ortogonalności – podstawa metody Galerkina. Generuje ten sam układ równań liniowych co MNK.

Aproksymacja funkcji w przestrzeni funkcyjnej

Niech V będzie *przestrzenią funkcyjną* rozpiętą na zbiorze *funkcji bazowych* $\psi_0, \ldots, \psi_N$,

$$V = \text{span}\,\{\psi_0, \ldots, \psi_N\}$$

Dowolną funkcję tej przestrzeni $u \in V$ można przedstawić jako kombinację liniową funkcji bazowych:

$$u = \sum_{j \in \mathcal{I}_s} c_j \psi_j, \quad \mathcal{I}_s = \{0, 1, \ldots, N\}$$

Uogólnienie MNK na przestrzenie funkcyjne

Tak jak dla przestrzeni wektorowych, minimalizujemy normę błędu E, ze względu na współczynniki c_j, $j \in \mathcal{I}_s$:

$$E = (e, e) = (f - u, f - u) = \left(f(x) - \sum_{j \in \mathcal{I}_s} c_j \psi_j(x), f(x) - \sum_{j \in \mathcal{I}_s} c_j \psi_j(x) \right)$$

$$\frac{\partial E}{\partial c_i} = 0, \quad i = \in \mathcal{I}_s$$

Czym jest iloczyn skalarny jeśli ψ_i jest funkcją?

$$(f, g) = \int_\Omega f(x) g(x) \, dx$$

(iloczyn skalarny funkcji cg jako uogólnienie il. skalarnego funkcji dyskretnych $(\boldsymbol{u}, \boldsymbol{v}) = \sum_j u_j v_j$)

Szczegóły MNK

$$E(c_0, \ldots, c_N) = (e, e) = (f - u, f - u)$$
$$= (f, f) - 2 \sum_{j \in \mathcal{I}_s} c_j (f, \psi_i) + \sum_{p \in \mathcal{I}_s} \sum_{q \in \mathcal{I}_s} c_p c_q (\psi_p, \psi_q)$$

$$\frac{\partial E}{\partial c_i} = 0, \quad i = \in \mathcal{I}_s$$

Obliczenia identyczne jak dla przypadku wektorowego -> w rezultacie otrzymujemy układ równań liniowych

$$\sum_{j \in \mathcal{I}_s}^{N} A_{i,j} c_j = b_i, \ i \in \mathcal{I}_s, \quad A_{i,j} = (\psi_i, \psi_j), \ b_i = (f, \psi_i)$$

Metoda Galerkina

Jak poprzednio minimalizacja (e, e) jest równoważna

$$(e, \psi_i) = 0, \quad i \in \mathcal{I}_s$$

co z kolei jest równoważne

$$(e, v) = 0, \quad \forall v \in V$$

Równoważność wzorów jak dla przestrzeni wektorowych.

Równoważność wzorów jak dla wyprowadzenia przy pomocy MNK.

Przykład: aproksymacja paraboli funkcją liniową

Problem

Dla zadanej funkcji $f(x) = 10(x-1)^2 - 1$ znaleźć jej przybliżenie funkcją liniową.

$$V = \text{span}\,\{1, x\}$$

czyli $\psi_0(x) = 1$, $\psi_1(x) = x$ oraz $N = 1$. Szukane

$$u = c_0\psi_0(x) + c_1\psi_1(x) = c_0 + c_1 x$$

Przykład: aproksymacja paraboli funkcją liniową

$$A_{0,0} = (\psi_0, \psi_0) = \int_1^2 1 \cdot 1 \, dx = 1$$

$$A_{0,1} = (\psi_0, \psi_1) = \int_1^2 1 \cdot x \, dx = 3/2$$

$$A_{1,0} = A_{0,1} = 3/2$$

$$A_{1,1} = (\psi_1, \psi_1) = \int_1^2 x \cdot x \, dx = 7/3$$

$$b_1 = (f, \psi_0) = \int_1^2 (10(x-1)^2 - 1) \cdot 1 \, dx = 7/3$$

$$b_2 = (f, \psi_1) = \int_1^2 (10(x-1)^2 - 1) \cdot x \, dx = 13/3$$

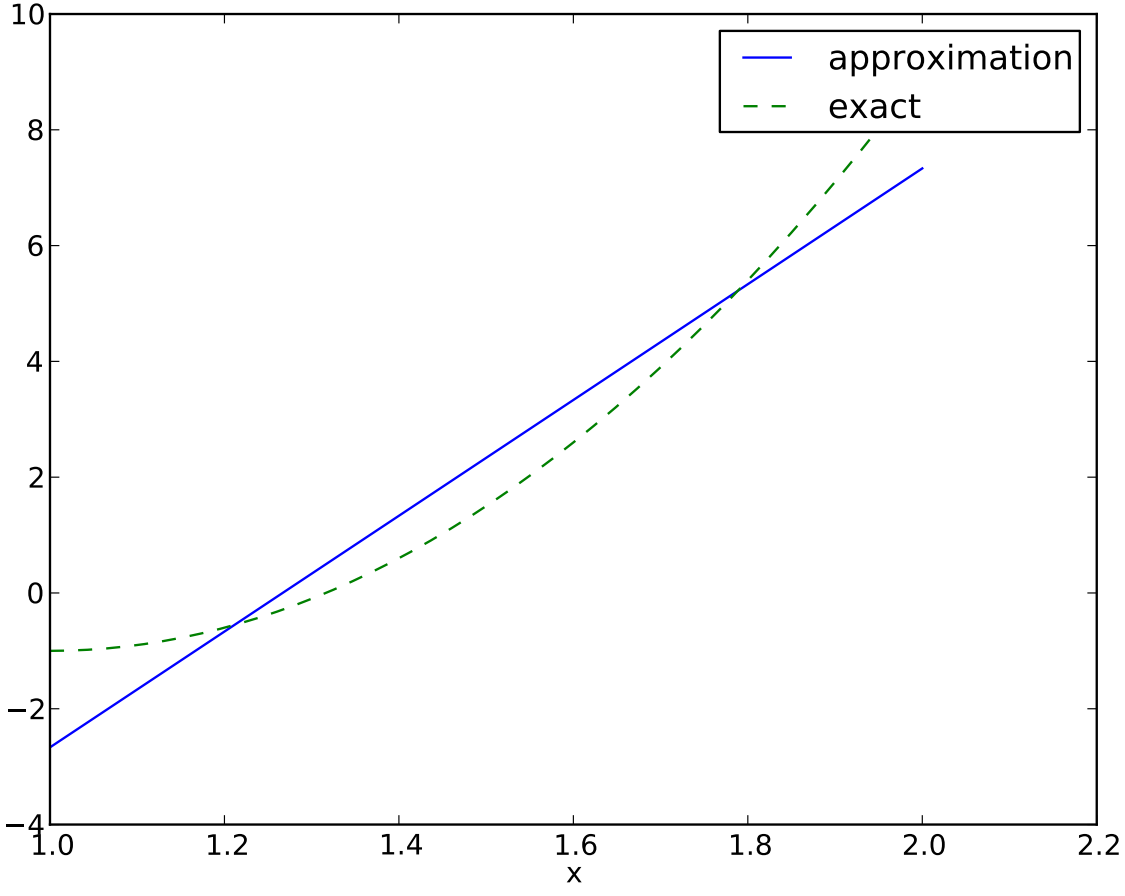
Rozwiązanie układu równań 2x2:

$$c_0 = -38/3, \quad c_1 = 10, \quad u(x) = 10x - \frac{38}{3}$$

Przykład: aproksymacja paraboli funkcją liniową

$$\begin{bmatrix} 1 & \frac{3}{2} \\ \frac{3}{2} & \frac{7}{3} \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} \frac{7}{3} \\ \frac{13}{3} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} -38/3 \\ 10 \end{bmatrix}$$

$$u(x) = 10x - 12\frac{2}{3}$$



Symboliczna realizacja algorytmu MNK

Problem: napisać program/funkcję, który przeprowadzi obliczenia (obliczenie całek i rozwiązanie układu równań liniowych) i zwróci rozwiązanie postaci n $u(x) = \sum_j c_j \psi_j(x)$.

Niech

- $f(x)$ będzie dane przez funkcję `sympy` oznaczoną symbolem `f` (funkcję zmiennej (symbolu) `x`)
- `psi` będzie listą funkcji $\{\psi_i\}_{i \in \mathcal{I}_s}$,
- `Omega` będzie dwuelementową krotką/listą zawierającą początek i koniec przedziału Ω

MNK symbolicznie: podejście nr 1

```
import sympy as sym

def least_squares(f, psi, Omega):
    N = len(psi) - 1
    A = sym.zeros((N+1, N+1))
    b = sym.zeros((N+1, 1))
    x = sym.Symbol('x')
    for i in range(N+1):
        for j in range(i, N+1):
            A[i,j] = sym.integrate(psi[i]*psi[j],
                                   (x, Omega[0], Omega[1]))
            A[j,i] = A[i,j]
        b[i,0] = sym.integrate(psi[i]*f, (x, Omega[0], Omega[1]))
    c = A.LUsolve(b)
    u = 0
    for i in range(len(psi)):
        u += c[i,0]*psi[i]
    return u, c
```

Spostrzeżenie: macierz układu jest symetryczna, dzięki czemu można zoptymalizować proces wyznaczania jej elementów

- Może się zdażyć, że obliczanie całki się nie powiedzie (skomplikowana funkcja `f`) , `sym.integrate` zwróci wtedy obiekt typu `sym.Integral`. Ulepszenie kodu: sprawdzenie czy takie zdarzenie wystąpiło i ew. obliczenia numeryczne.
- Ulepszenie 2: Dodatkowa flaga przy pomocy, której użytkownik może zdecydować bezpośrednio jaki rodzaj całkowania (symboliczny czy numeryczny) ma zostać wykorzystany.

MNK symbolicznie: podejście nr 2

```
def least_squares(f, psi, Omega, symbolic=True):
    ...
    for i in range(N+1):
        for j in range(i, N+1):
            integrand = psi[i]*psi[j]
            if symbolic:
                I = sym.integrate(integrand, (x, Omega[0], Omega[1]))
            if not symbolic or isinstance(I, sym.Integral):
                # Could not integrate symbolically,
                # fall back on numerical integration
                integrand = sym.lambdify([x], integrand)
                I = sym.mpmath.quad(integrand, [Omega[0], Omega[1]])
            A[i,j] = A[j,i] = I

        integrand = psi[i]*f
        if symbolic:
            I = sym.integrate(integrand, (x, Omega[0], Omega[1]))
        if not symbolic or isinstance(I, sym.Integral):
            integrand = sym.lambdify([x], integrand)
            I = sym.mpmath.quad(integrand, [Omega[0], Omega[1]])
        b[i,0] = I
    ...
```

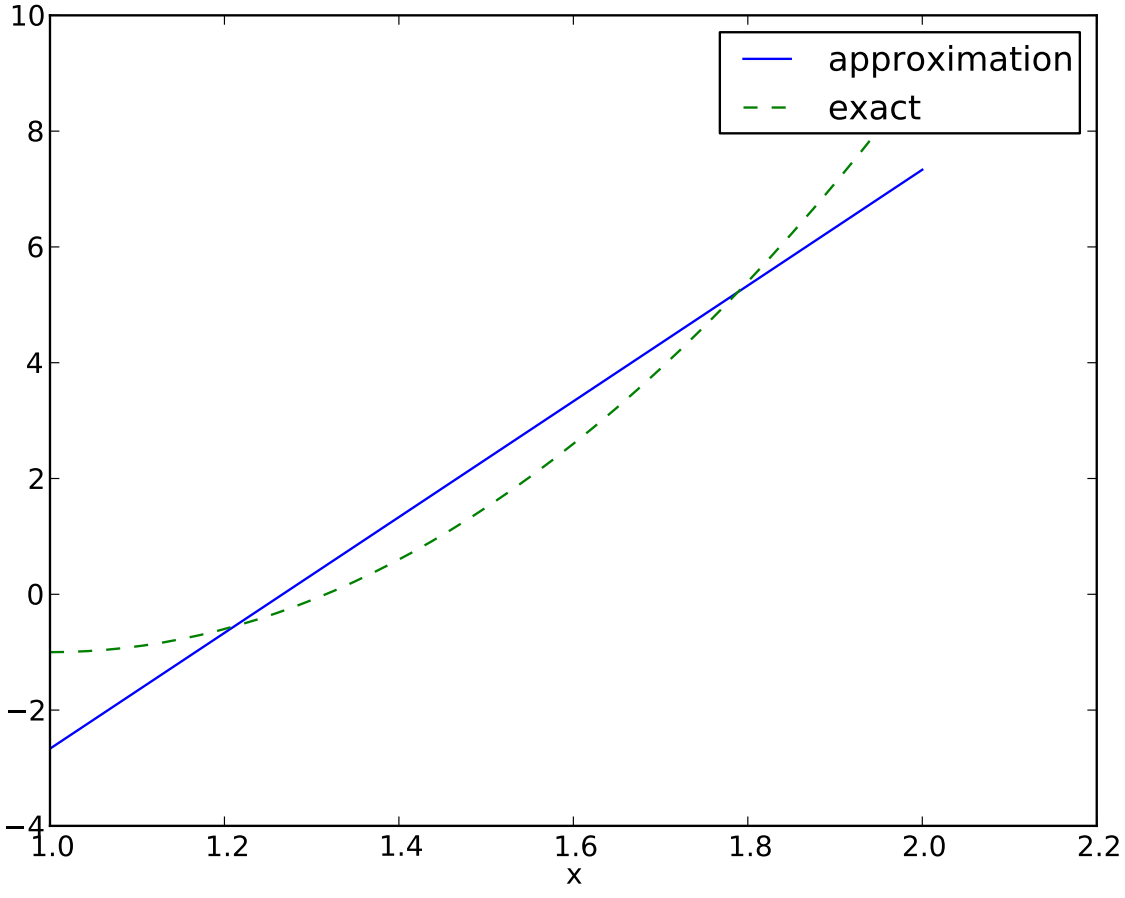
Prezentacja rozwiązania

Graficzne porównanie f i u:

```
def comparison_plot(f, u, Omega, filename='tmp.pdf'):
    x = sym.Symbol('x')
    # Turn f and u to ordinary Python functions
    f = sym.lambdify([x], f, modules="numpy")
    u = sym.lambdify([x], u, modules="numpy")
    resolution = 401  # no of points in plot
    xcoor  = linspace(Omega[0], Omega[1], resolution)
    exact  = f(xcoor)
    approx = u(xcoor)
    plot(xcoor, approx)
    hold('on')
    plot(xcoor, exact)
    legend(['approximation', 'exact'])
    savefig(filename)
```

Zastosowanie kodu

```
>>> from approx1D import *
>>> x = sym.Symbol('x')
>>> f = 10*(x-1)**2-1
>>> u, c = least_squares(f=f, psi=[1, x], Omega=[1, 2])
>>> comparison_plot(f, u, Omega=[1, 2])
```



Przypadek aproksymacji funkcji $f \in V$

- Rozszerzmy zbiór funkcji bazowych przestrezni V o funkcję $\psi_2 = x^2$, wciąż poszukując przybliżenia dla funkcji $f = 10(x-1)^2 - 1$ w przestrzeni V
- -> przybliżenie paraboli pewną parabolą ...
- Rozwiązanie odwzoruje f ściśle!

```
>>> from approx1D import *
>>> x = sym.Symbol('x')
>>> f = 10*(x-1)**2-1
>>> u, c = least_squares(f=f, psi=[1, x, x**2], Omega=[1, 2])
>>> print u
10*x**2 - 20*x + 9
>>> print sym.expand(f)
10*x**2 - 20*x + 9
```

Rozwiązanie przybliżone $\equiv$ rozwiązanie dokładne, jeśli $f \in V$!

Uogólnienie: Przypadek aproksymacji funkcji $f \in V$

- A co jeśli baza to $\psi_i(x) = x^i$ dla $i = 0, \ldots, N = 40$?
- Wynik funkcji `least_squares`: dla $i > 2$, $c_i = 0$

Wniosek ogólny:

Jeśli $f \in V$, MNK oraz metoda Galerkina zwrócą $u = f$.

Dlaczego dla $f \in V$ aproksymacja jest bezbłędna? Dowód:

Jeśli $f \in V$, wtedy $f = \sum_{j \in \mathcal{I}_s} d_j \psi_j$, dla pewnego $\{d_i\}_{i \in \mathcal{I}_s}$. Wtedy

$$b_i = (f, \psi_i) = \sum_{j \in \mathcal{I}_s} d_j (\psi_j, \psi_i) = \sum_{j \in \mathcal{I}_s} d_j A_{i,j}$$

a URL $\sum_j A_{i,j} c_j = b_i$, $i \in \mathcal{I}_s$, przedstawia sie:

$$\sum_{j \in \mathcal{I}_s} c_j A_{i,j} = \sum_{j \in \mathcal{I}_s} d_j A_{i,j}, \quad i \in \mathcal{I}_s$$

co oznacza, że $c_i = d_i$ dla $i \in \mathcal{I}_s$, czyli u jest tożsame z f.

Skończona precyzja obliczeń numerycznych

Poprzednie wnioski -> teoria i obliczenia symboliczne . . .

Co w przypadku obliczeń numerycznych? -> (rozwiązanie URL macierzami liczb zmiennoprzecinkowych)

f to wciąż funkcja kwadratowa przybliżana przez

$$u(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3 + \cdots + c_N x^N$$

Oczekiwane: $c_2 = c_3 = \cdots = c_N = 0$, skoro $f \in V$ oznacza $u = f$.

A naprawdę?

Skończona prezycja obliczeń numerycznych – wyniki

teoria	sympy	numpy32	numpy64
9	9.62	5.57	8.98
-20	-23.39	-7.65	-19.93
10	17.74	-4.50	9.96
0	-9.19	4.13	-0.26
0	5.25	2.99	0.72
0	0.18	-1.21	-0.93
0	-2.48	-0.41	0.73
0	1.81	-0.013	-0.36
0	-0.66	0.08	0.11
0	0.12	0.04	-0.02
0	-0.001	-0.02	0.002

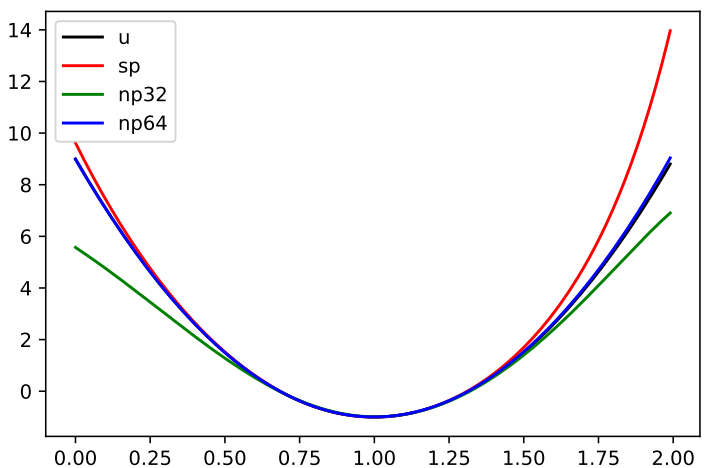
- Kolumna 2: `matrix` oraz `lu_solve` z biblioteki `sympy.mpmath.fp`
- Kolumna 3: `numpy` 4B liczby zmiennoprzecinkowe
- Kolumna 4: `numpy` 8B liczby zmiennoprzecinkowe

Złe uwarunkowanie URL - "liniowa zależność" w bazie

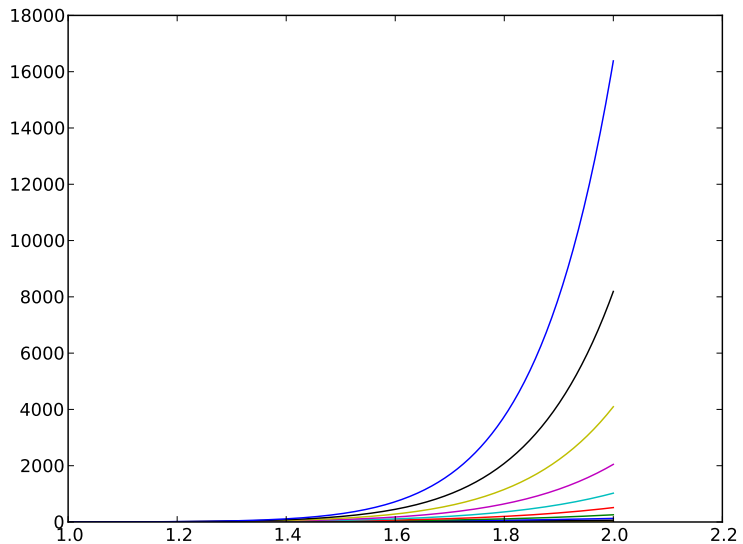
- Znaczne błędy zaokrągleń rozwiązania numerycznego (!)
- Jednocześnie "na oko" (graficzne) rozwiązanie wygląda w porządku (!)

Źródło kłopotów: funkcje x^i dla bardzo dużych i stają się praktycznie liniowo zależne

4 rozwiązania zadania przybliżenia paraboli



Wykresy funkcji x^i dla $i = 0 \ldots 14$



Złe uwarunkowanie URL: wnioski

- Prawie liniowa zależność funkcji bazowych skutkuje bliskoosobliwymi macierzami
- macierz prawie osobliwa $\equiv$ *macierz źle uwarunkowana* -> problemy w trakcie m.elim. Gaussa
- Baza wielomianów $1, x, x^2, x^3, x^4, \ldots$ to "nienajszczęśliwszy" wybór
- Istnieją lepsze bazy (nawet wielomianowe), ale im bardziej ortogonalne te bazy są, tym lepiej ($(\psi_i, \psi_j) \approx 0$)

Aproksymacja szeregami Fouriera; problem and code

Aproksymacja funkcji f szeregiem Fouriera

$$u(x) = \sum_i a_i \sin i\pi x = \sum_{j=0}^{N} c_j \sin((j+1)\pi x)$$

to tylko "zmiana bazy":

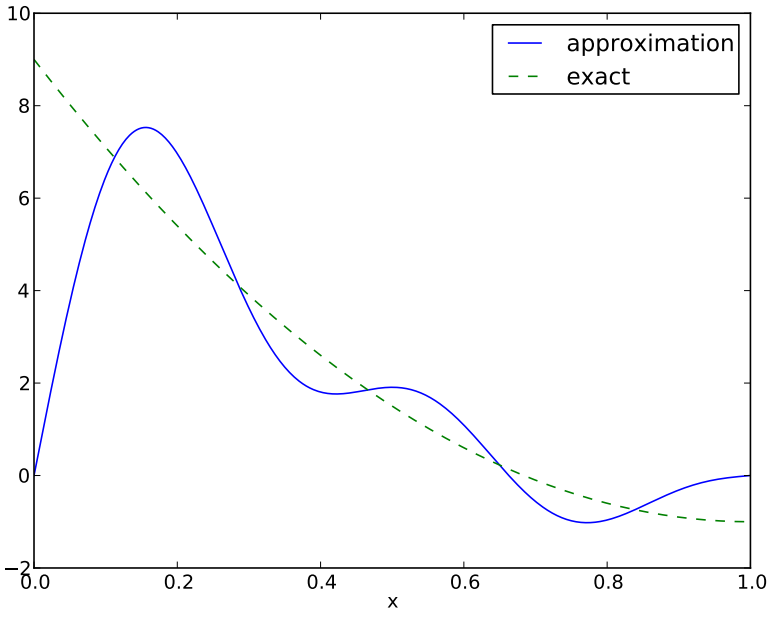
$$V = \operatorname{span}\{\sin \pi x, \sin 2\pi x, \ldots, \sin(N+1)\pi x\}$$

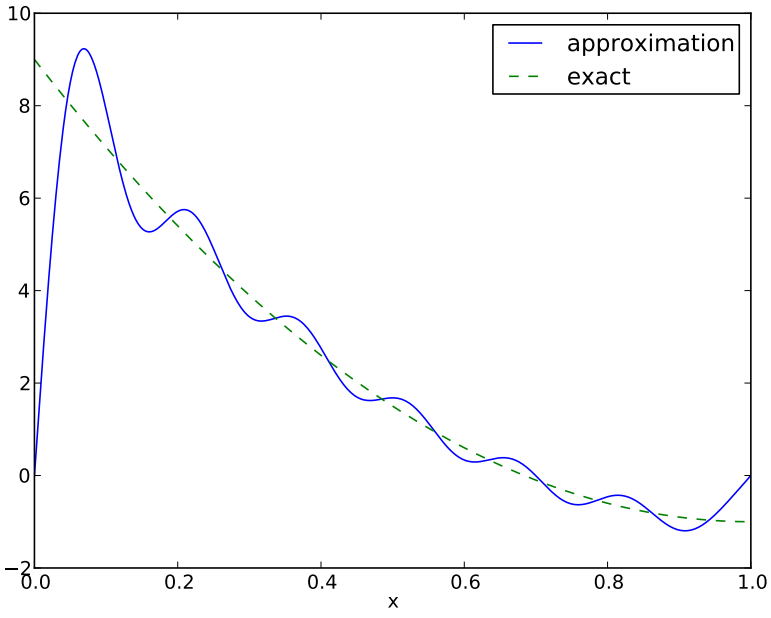
Obliczenia z wykorzystaniem funkcji `least_squares`:

```
N = 3
from sympy import sin, pi
psi = [sin(pi*(i+1)*x) for i in range(N+1)]
f = 10*(x-1)**2 - 1
Omega = [0, 1]
u, c = least_squares(f, psi, Omega)
comparison_plot(f, u, Omega)
```

Aproksymacja szeregami Fouriera; wykres

L: $N = 3$, P: $N = 11$:





Problem:

Dla każdej f.bazowej jest $\psi_i(0) = 0$ przez co $u(0) = 0 \neq f(0) = 9$. Podobna sytuacja dla $x = 1$. Wartości u na brzegach będą zawsze niepoprawne!

Aproksymacja szeregami Fouriera; ulepszenie

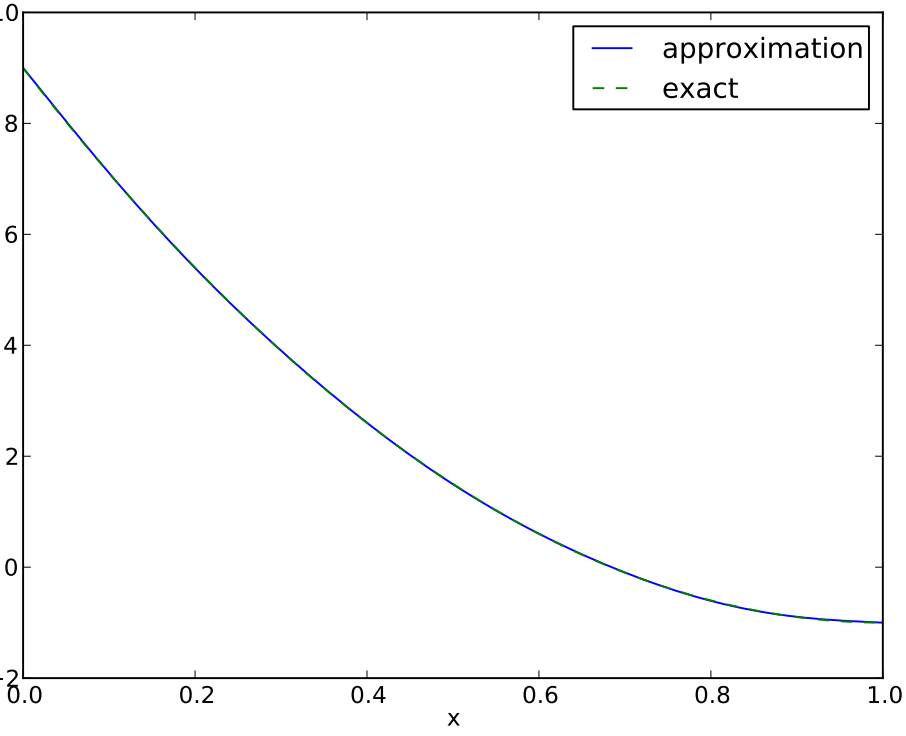
- Znaczna poprawa aproksymacja dla $N = 11$ wyrazów, pomimo niepożądanych rozbieżności w $x = 0$ i $x = 1$
- Możliwe rozwiązanie: dodać składową, która pozwoli na odwzorowanie właściwych wartości na brzegu

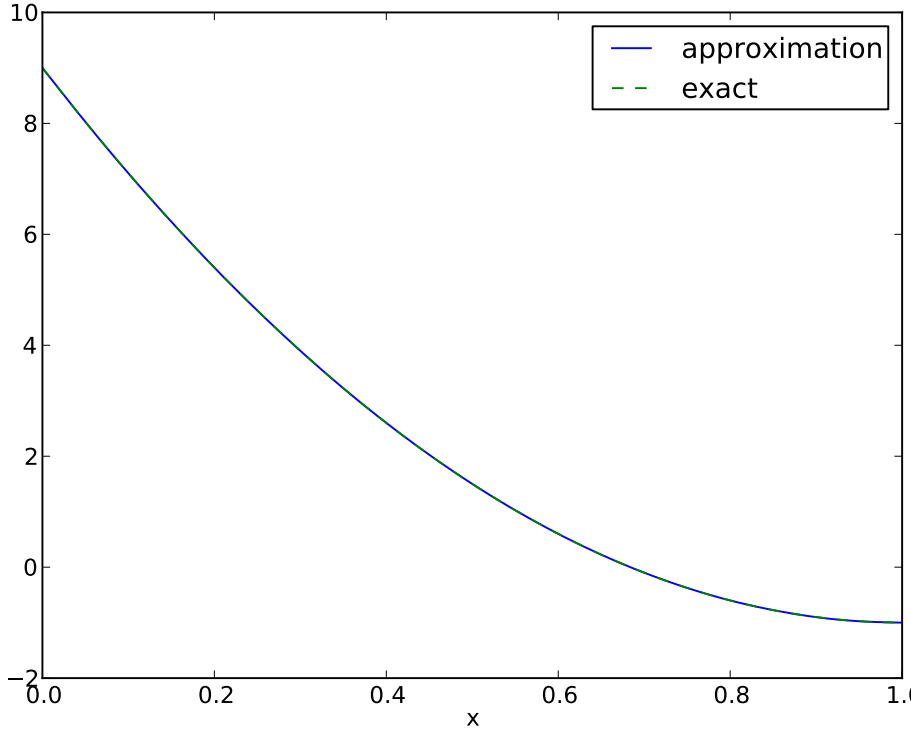
$$u(x) = f(0)(1 - x) + xf(1) + \sum_{j \in \mathcal{I}_s} c_j \psi_j(x)$$

Dodatkowy wyraz nie tylko zapewnia $u(0) = f(0)$ oraz $u(1) = f(1)$, ale także zaskakująco dobrze poprawia jakość aproksymacji!

Aproksymacja szeregami Fouriera; wyniki

$N = 3$ vs $N = 11$:





Bazy funkcji ortogonalnych

Zalety wyboru funkcji sinus jako funkcji bazowych:

- funkcje bazowe są parami ortogonalne: $(\psi_i, \psi_j) = 0$ (jedynie $(\psi_i, \psi_j) \neq 0$)) dzięki czemu
- macierz $A_{i,j}$ jest diagonalna, dzięki czemu
- nie ma potrzeby rozwiązywać URL! Rozwiązanie sprowadza się do obliczenia: $c_i = 2 \int_0^1 f(x) \sin((i+1)\pi x) dx$
- wynik: rozwinięcie funkcji f w szereg Fouriera

W ogólnym przypadku, dla baz ortogonalnych, $A_{i,j}$ jest macierzą diagonalną, a nieznane współczynniki c_i można łatwo obliczyć:

$$c_i = \frac{b_i}{A_{i,i}} = \frac{(f, \psi_i)}{(\psi_i, \psi_i)}$$

Implementacja MNK dla ortogonalnych funkcji bazowych

```
def least_squares_orth(f, psi, Omega):
    N = len(psi) - 1
    A = [0]*(N+1)
    b = [0]*(N+1)
    x = sym.Symbol('x')
    for i in range(N+1):
        A[i] = sym.integrate(psi[i]**2, (x, Omega[0], Omega[1]))
        b[i] = sym.integrate(psi[i]*f,  (x, Omega[0], Omega[1]))
    c = [b[i]/A[i] for i in range(len(b))]
    u = 0
    for i in range(len(psi)):
        u += c[i]*psi[i]
    return u, c
```

Implementacja MNK dla ortogonalnych funkcji bazowych: całkowanie symboliczne i numeryczne

- Uwzględnienie parametru sterującego wyborem rodzaju całkowania (argument `symbolic`).
- W przypadku gdy całkowanie symboliczne zawiedzie (`sym.integrate` zwraca `sym.Integral`), obliczenia wykonywane numerycznie (w przypadku funkcji sinus nie powinno być problemów z symbolicznym obliczeniem $\int_\Omega \varphi_i^2 dx$)

```
def least_squares_orth(f, psi, Omega, symbolic=True):
    ...
    for i in range(N+1):
        # Diagonal matrix term
        A[i] = sym.integrate(psi[i]**2, (x, Omega[0], Omega[1]))

        # Right-hand side term
        integrand = psi[i]*f
        if symbolic:
            I = sym.integrate(integrand,  (x, Omega[0], Omega[1]))
        if not symbolic or isinstance(I, sym.Integral):
            print 'numerical integration of', integrand
            integrand = sym.lambdify([x], integrand)
            I = sym.mpmath.quad(integrand, [Omega[0], Omega[1]])
        b[i] = I
    ...
```

Metoda kolokacji (interpolacji); idea i teoria

Inny sposób znalezienia przybliżenia $f(x)$ przez $u(x) = \sum_j c_j \psi_j$:

- Wymuszamy $u(x_i) = f(x_i)$ w pewnych wybranych punktach $\{x_i\}_{i \in \mathcal{I}_s}$ (punktach *kolokacji*)
- *u interpoluje f*
- metoda znana jako metoda *kolokacji* (*interpolacji*)

$$u(x_i) = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x_i) = f(x_i) \quad i \in \mathcal{I}_s, N$$

Współczynniki wygenerowanego układu równań to po prostu wartości funkcji, nie ma potrzeby całkowania!

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \quad i \in \mathcal{I}_s \tag{5}$$

$$A_{i,j} = \psi_j(x_i) \tag{6}$$

$$b_i = f(x_i) \tag{7}$$

W ogólnym przypadku macierz wynikowa niesymetryczna: $\psi_i(x_i) \neq \psi_i(x_i)$

Metoda kolokacji – implementacja

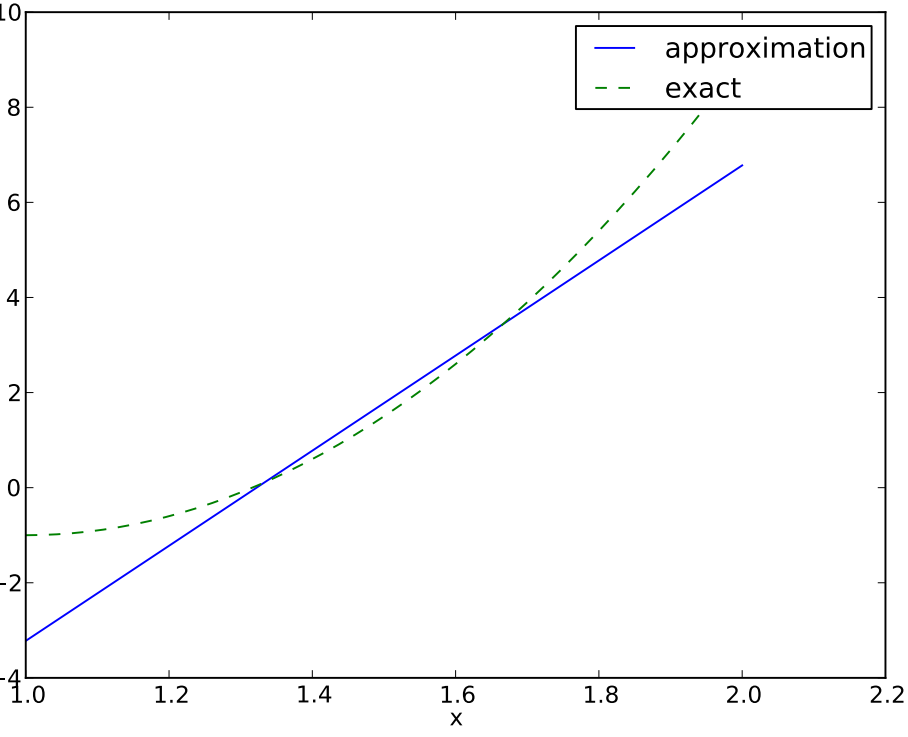
Zmienna points przechowuje punkty kolokacji

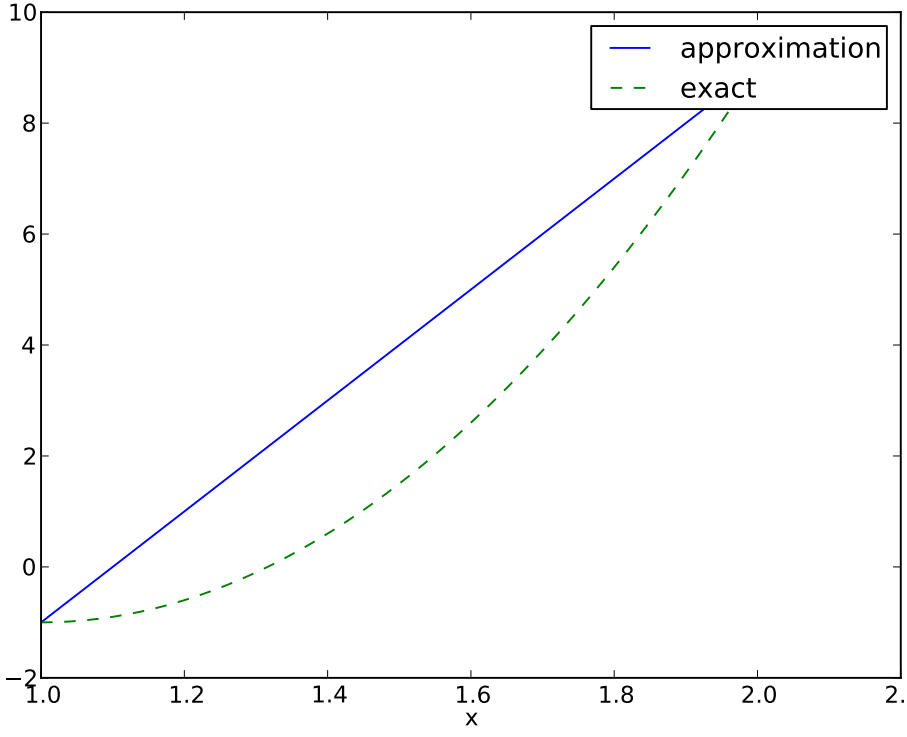
```
def interpolation(f, psi, points):
    N = len(psi) - 1
    A = sym.zeros((N+1, N+1))
    b = sym.zeros((N+1, 1))
    x = sym.Symbol('x')
    # Turn psi and f into Python functions
    psi = [sym.lambdify([x], psi[i]) for i in range(N+1)]
    f = sym.lambdify([x], f)
    for i in range(N+1):
        for j in range(N+1):
            A[i,j] = psi[j](points[i])
        b[i,0] = f(points[i])
    c = A.LUsolve(b)
    u = 0
    for i in range(len(psi)):
        u += c[i,0]*psi[i](x)
    return u
```

Metoda kolokacji: przybliżenie paraboli funkcją liniową

- Problem: jak wybrać x_i?
- Wynik zależy od położenia punktów kolokacji!

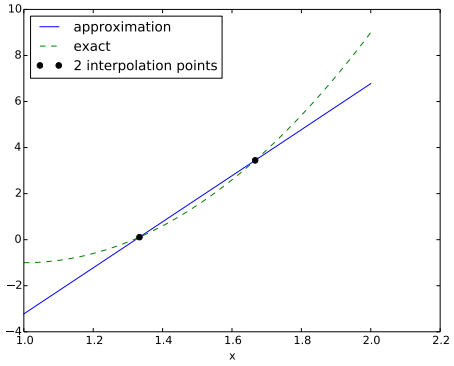
$(4/3, 5/3)$ vs $(1, 2)$:

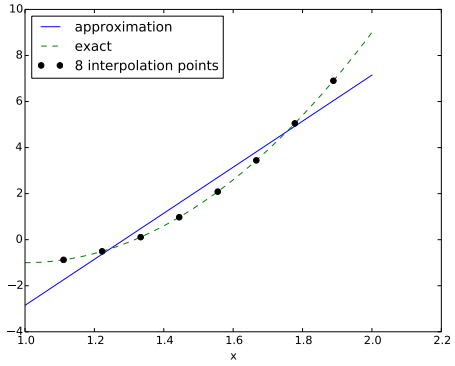


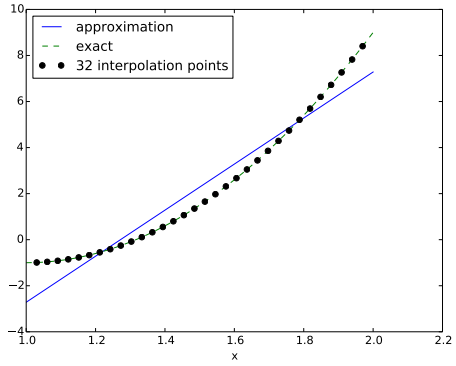


Regresja

- Idea: metoda kolokacji dla $m \gg N + 1$ punktów
- Problem: Więcej równań niż niewiadomych
- Znana np. ze statystyki *regresja*







Regresja – nadokreślone URL

$$u(x_i) = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x_i) = f(x_i), \quad i = 0, 1, \ldots, m$$

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \quad i = 0, 1, \ldots, m$$

$$A_{i,j} = \psi_j(x_i), \quad b_i = f(x_i)$$

Rozwiązywanie nadokreślonych URL przy pomocy MNK

- Jak rozwiązać $Ac = b$ jeśli jest więcej równań niż niewiadomych?
- Idea: Poszukiwanie rozwiązania minimalizującego $r = b - Ac$
- Rezultat: układ równań normalnych $A^T Ac = A^T b$
- Zapiszmy układ w postaci $Bc = d$
- $B = A^T A$ już kwadratowe: układ równań o rozmiarach $(N+1) \times (N+1)$

$$B_{i,j} = \sum_k A^T i, k A_{k,j} = \sum_k Ak, i A_{k,j} = \sum_{k=0}^{m} \psi_i(x_k)\psi_j(x_k)$$

$$d_i = \sum_k A^T_{i,k} b_k = \sum_k A_{k,i} b_k = \sum_{k=0}^{m} \psi_i(x_k) f(x_k)$$

Implementacja

```
def regression(f, psi, points):
    N = len(psi) - 1
    m = len(points)
    # Use numpy arrays and numerical computing
    B = np.zeros((N+1, N+1))
    d = np.zeros(N+1)
    # Wrap psi and f in Python functions rather than expressions
    # so that we can evaluate psi at points[i]
    x = sym.Symbol('x')
    psi_sym = psi  # save symbolic expression
    psi = [sym.lambdify([x], psi[i]) for i in range(N+1)]
    f = sym.lambdify([x], f)
    for i in range(N+1):
        for j in range(N+1):
            B[i,j] = 0
            for k in range(m+1):
                B[i,j] += psi[i](points[k])*psi[j](points[k])
        d[i] = 0
        for k in range(m+1):
            d[i] += psi[i](points[k])*f(points[k])
    c = np.linalg.solve(B, d)
    u = sum(c[i]*psi_sym[i] for i in range(N+1))
    return u, c
```

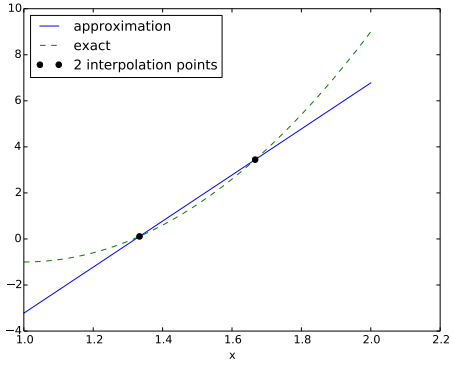
Przykład zastosowania – kod

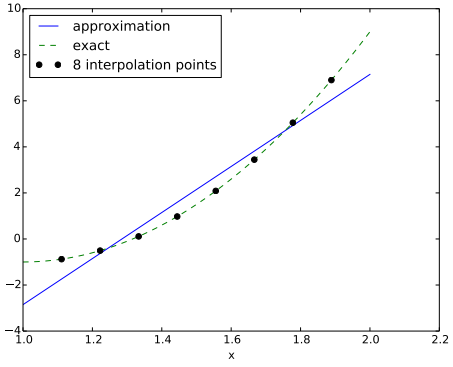
- Zadanie: Dokonać aproksymacji funkcji $f(x) = 10(x-1)^2 - 1$ na przedziale $\Omega = [1, 2]$ przy pomocy funkcji liniowej.

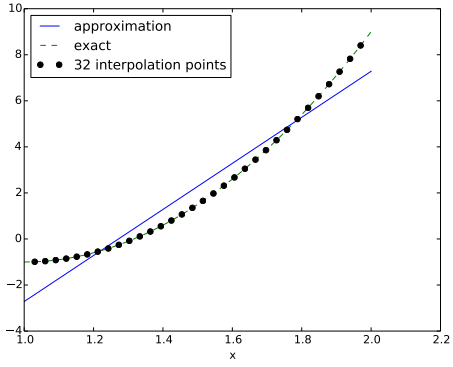
```
import sympy as sym
x = sym.Symbol('x')
f = 10*(x-1)**2 - 1
psi = [1, x]
Omega = [1, 2]
m_values = [2-1, 8-1, 64-1]
# Create m+3 points and use the inner m+1 points
for m in m_values:
    points = np.linspace(Omega[0], Omega[1], m+3)[1:-1]
    u, c = regression(f, psi, points)
    comparison_plot(f, u, Omega, points=points,
        points_legend='%d interpolation points' % (m+1))
```

Przykład zastosowania – wyniki

$$u(x) = 10x - 13.2, \quad 2 \text{ punkty}$$
$$u(x) = 10x - 12.7, \quad 8 \text{ punktów}$$
$$u(x) = 10x - 12.7, \quad 64 \text{ punkty}$$







Wielomiany Lagrange'a

Motywacja::

- Metoda kolokacji pozwala uniknąć całkowania
- Dla macierzy diagonalnej $A_{i,j} = \psi_j(x_i)$ rozwiązanie URL jest banalnie proste

Własność *wielomianów Lagrange'a* ψ_j:

$$\psi_i(x_j) = \delta_{ij}, \quad \delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$$

Zatem, $c_i = f(x_i)$ and

$$u(x) = \sum_{j \in \mathcal{I}_s} f(x_i)\psi_i(x)$$

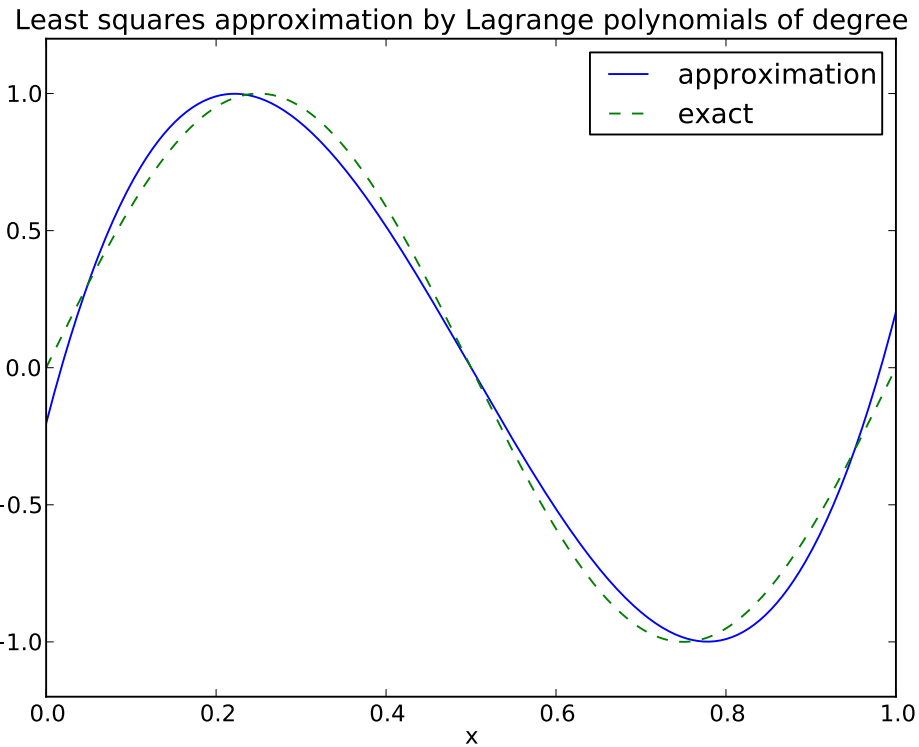
- Wielomiany Lagrange'a w połączeniu z metodą kolokacji są niezwykle wygodne
- Często stosowane w FEM

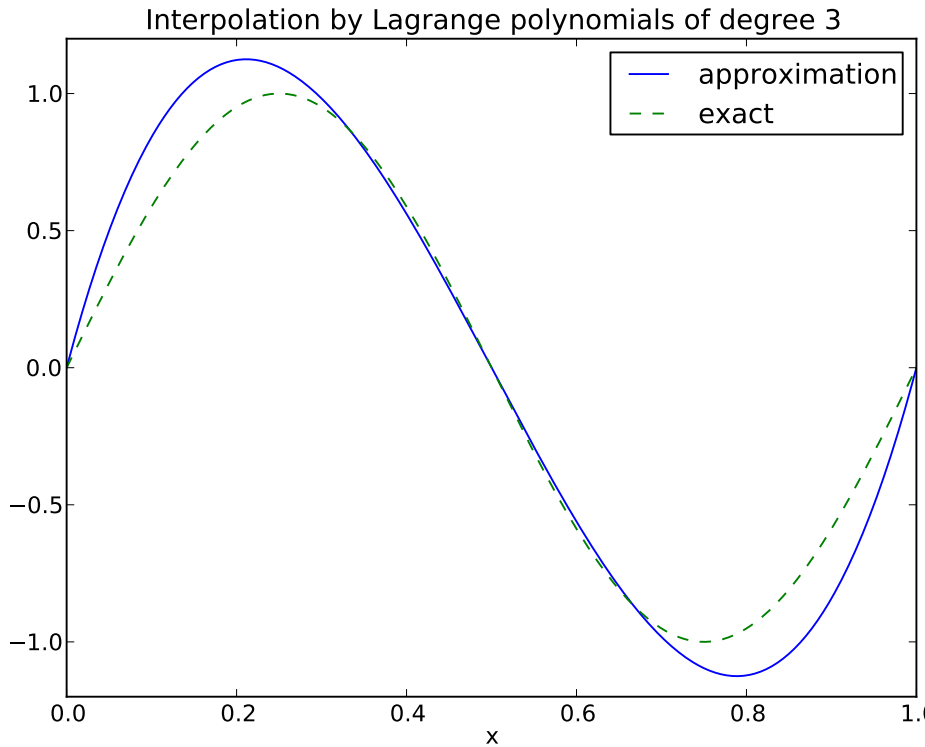
Wielomiany Lagrange'a – wzór i implementacja

$$\psi_i(x) = \prod_{j=0, j\neq i}^{N} \frac{x - x_j}{x_i - x_j} = \frac{x - x_0}{x_i - x_0} \cdots \frac{x - x_{i-1}}{x_i - x_{i-1}} \frac{x - x_{i+1}}{x_i - x_{i+1}} \cdots \frac{x - x_N}{x_i - x_N}$$

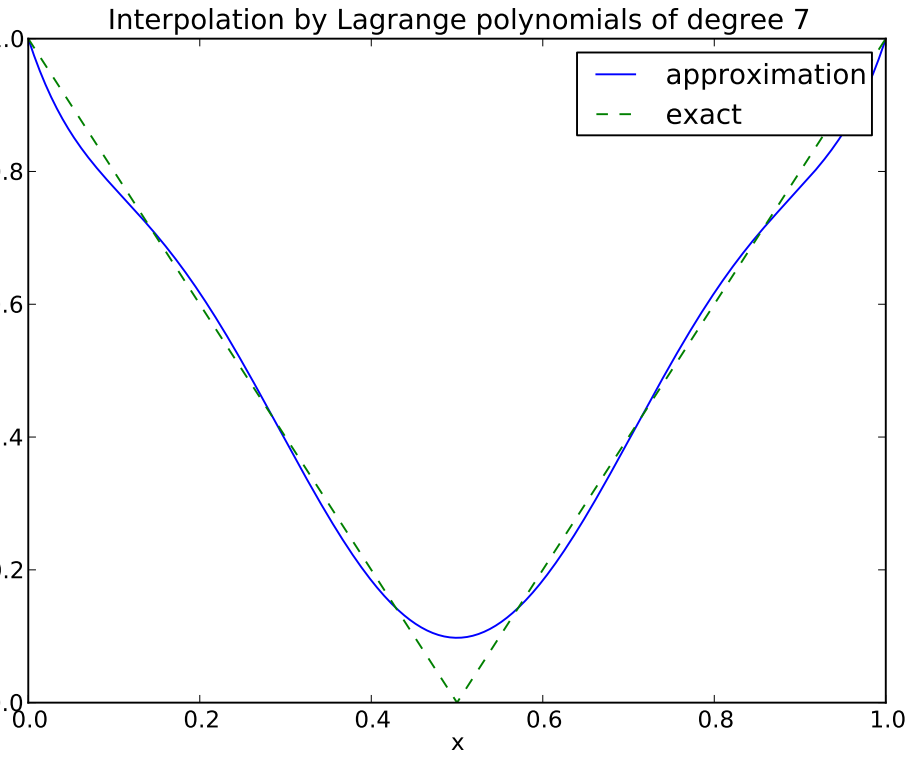
```
def Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p
```

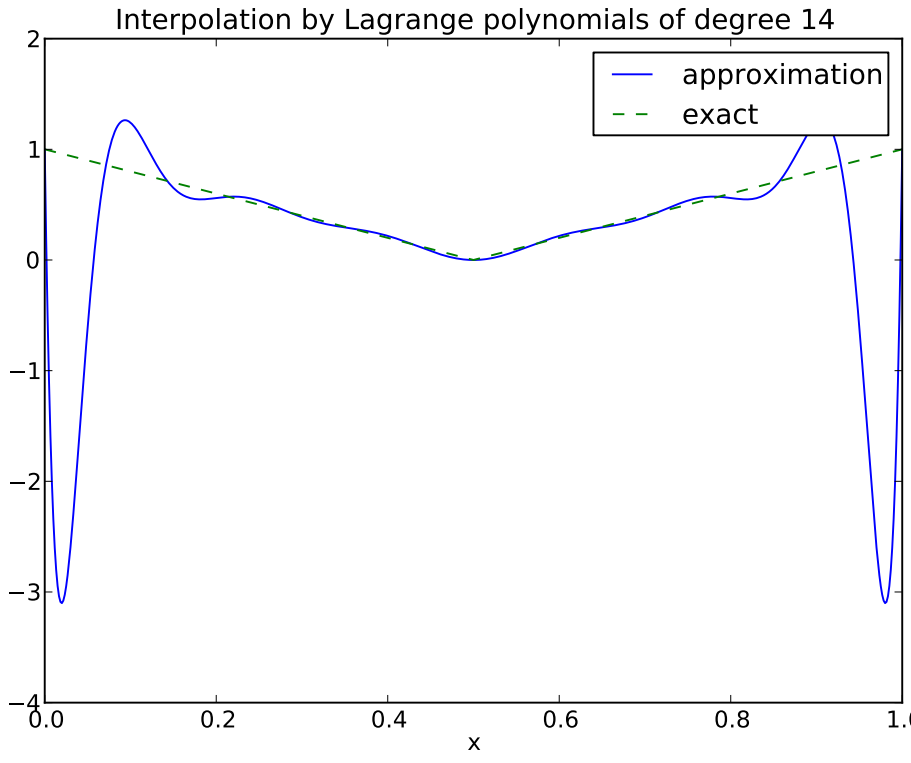
Wielomiany Lagrange'a – zachęcający przykład zastosowania





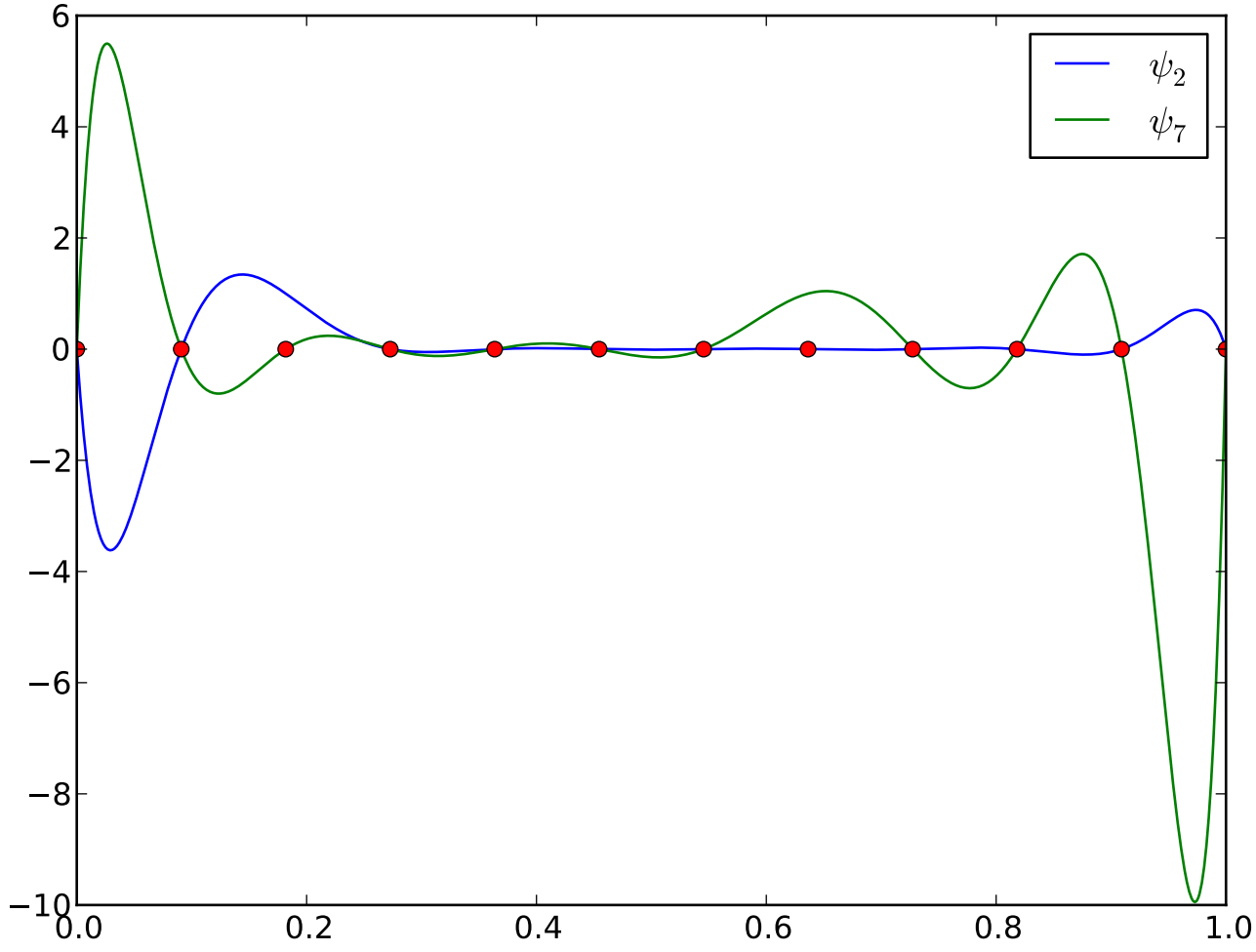
Wielomiany Lagrange'a – mniej zachęcający przykład zastosowania





Wielomiany Lagrange'a – efekt Runge'go

12 punktów, dwa wielomiany stopnia 11 (Uwaga!: $\psi_2(x_2) \neq 0$ i $\psi_7(x_7) \neq 0$



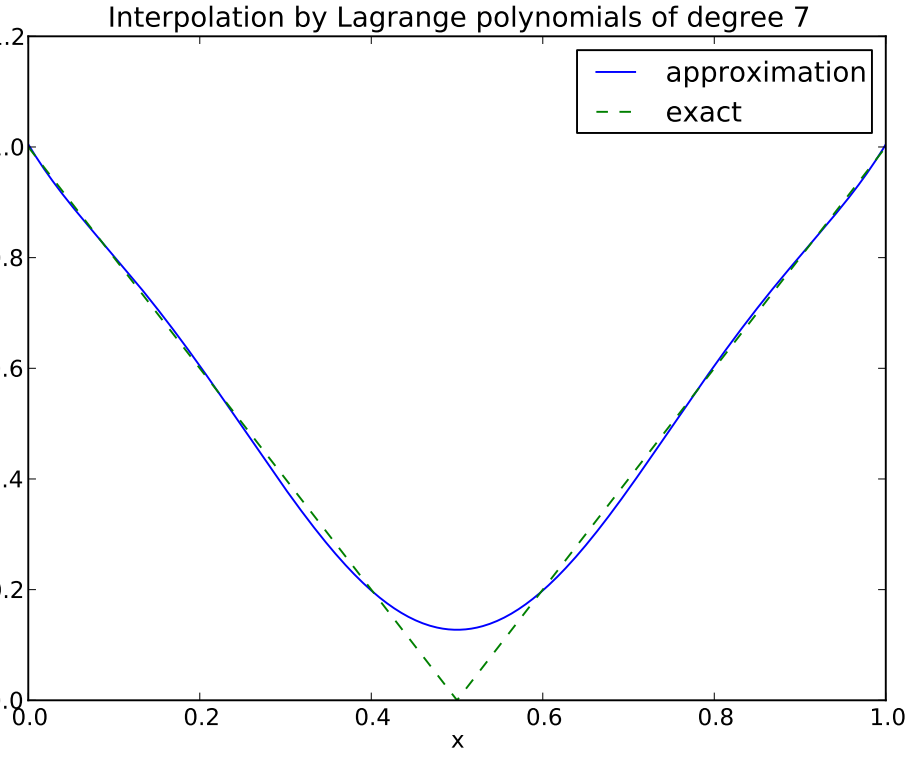
Wielomiany Lagrange'a: jak zapobiec oscylacjom?

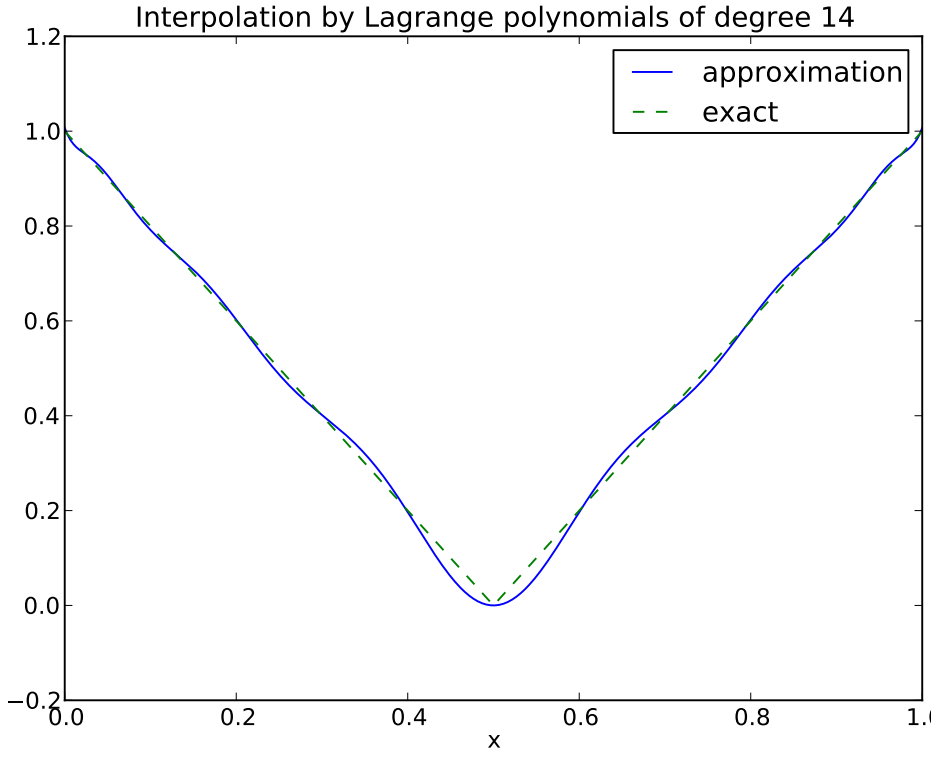
Odpowiedni dobór węzłów interpolacji – *węzły Czebyszewa*:

$$x_i = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\cos\left(\frac{2i+1}{2(N+1)}\pi\right), \quad i = 0\ldots, N$$

na przedziale $[a, b]$.

Wielomiany Lagrange'a + węzły Czebyszewa



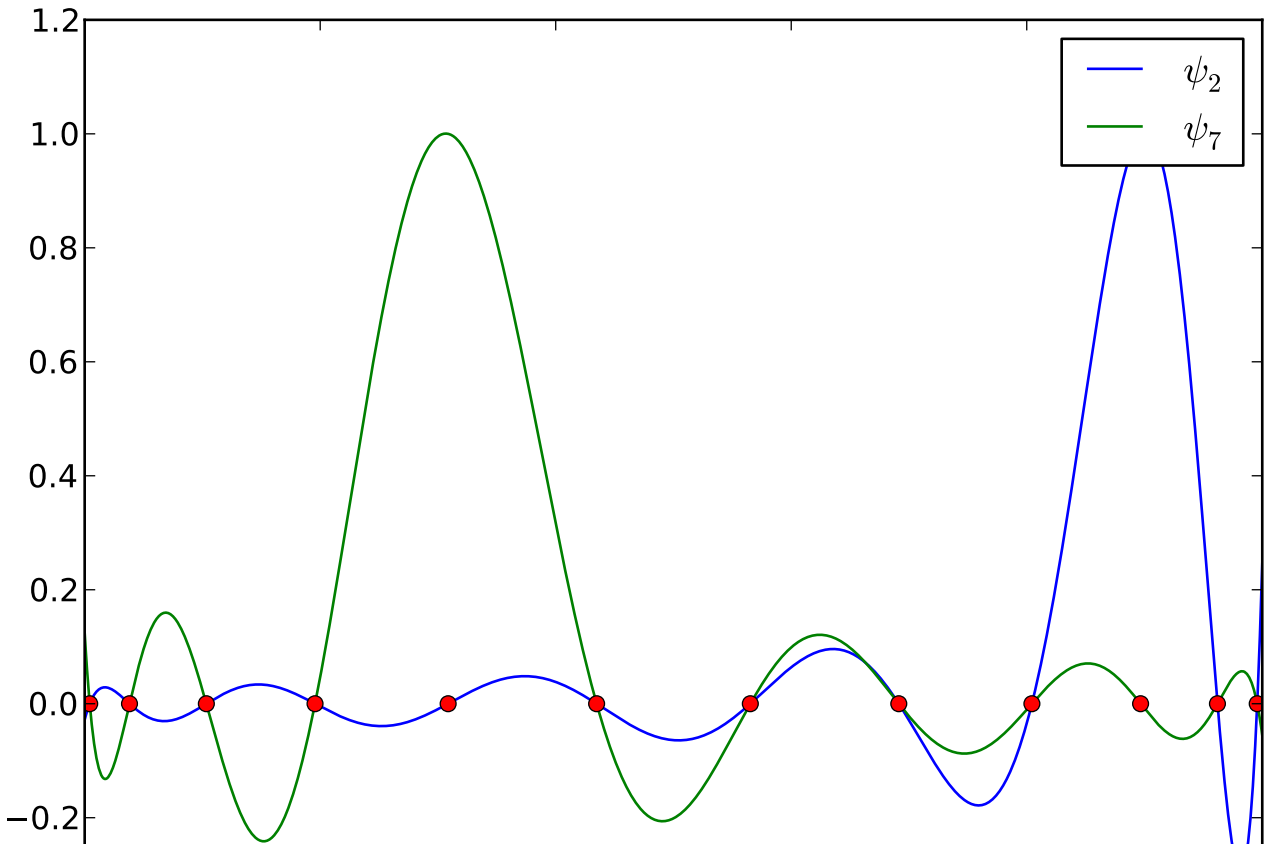


Wielomiany Lagrange'a + węzły Czebyszewa

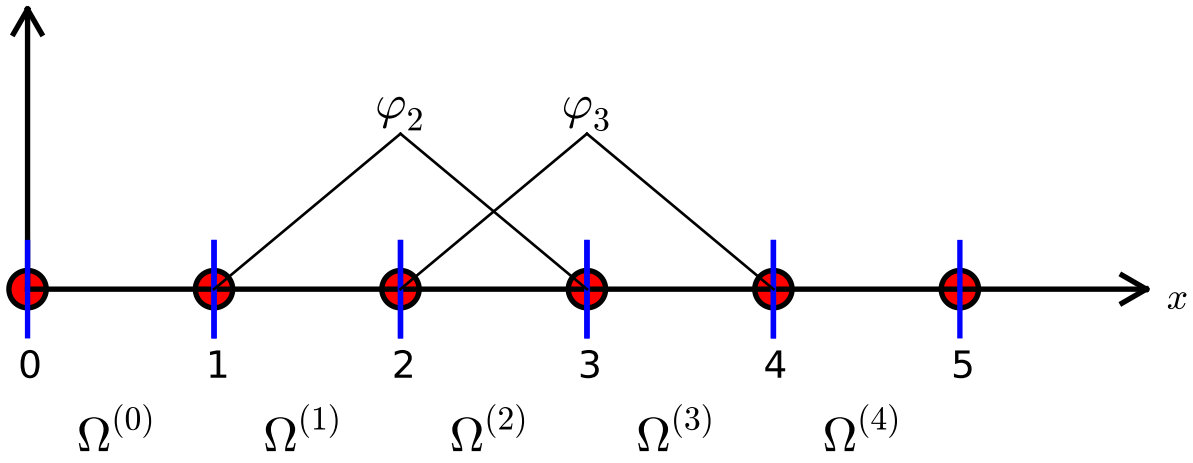
12 punktów, dwa wielomiany stopnia 11.

Uwaga!: Tym razem węzły są inaczej rozmieszczone!

Mniej oscylacyjny charakter wielomianów w porównaniu do węzłów równomiernie rozmieszczonych.

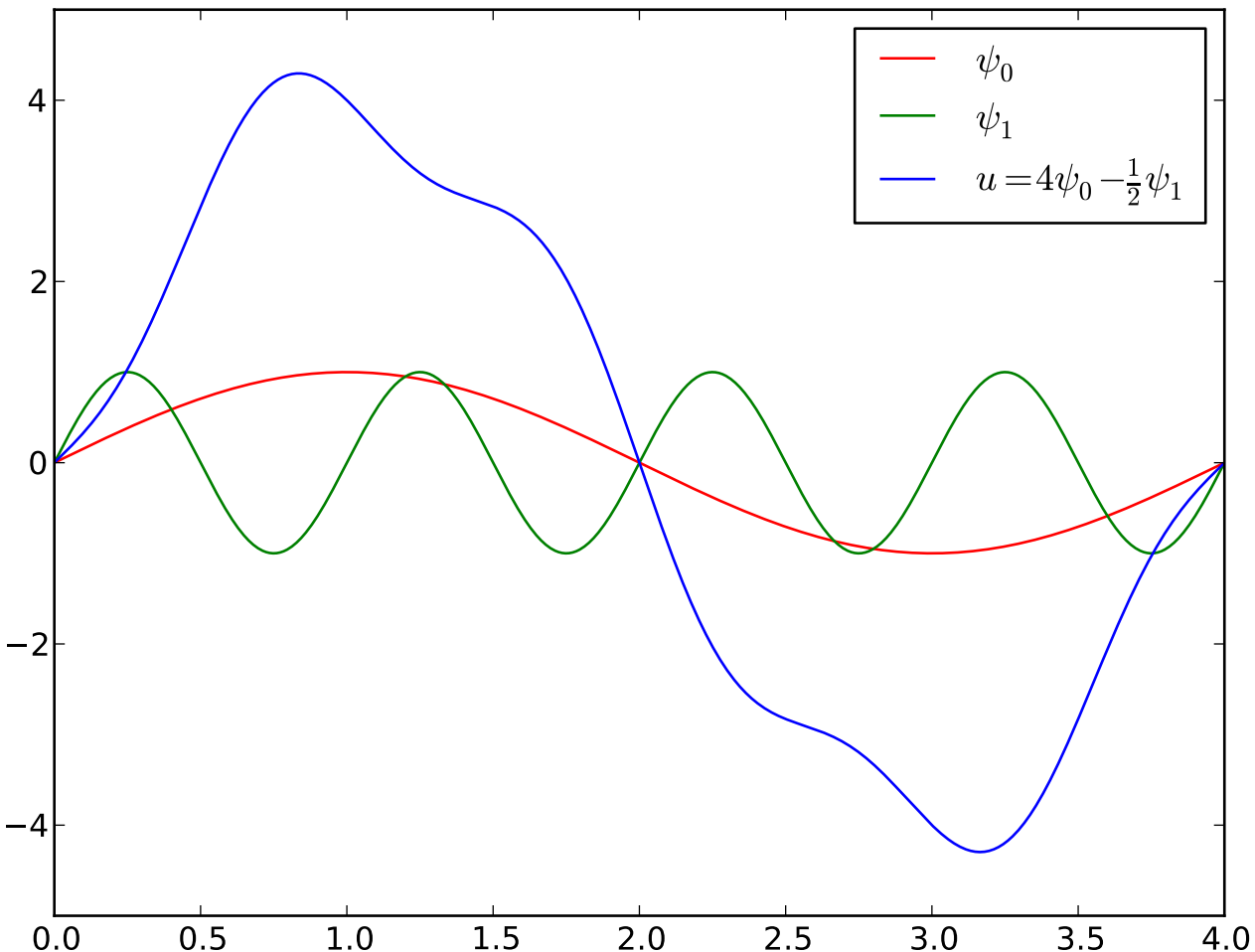


Funkcje bazowe elementów skończonych



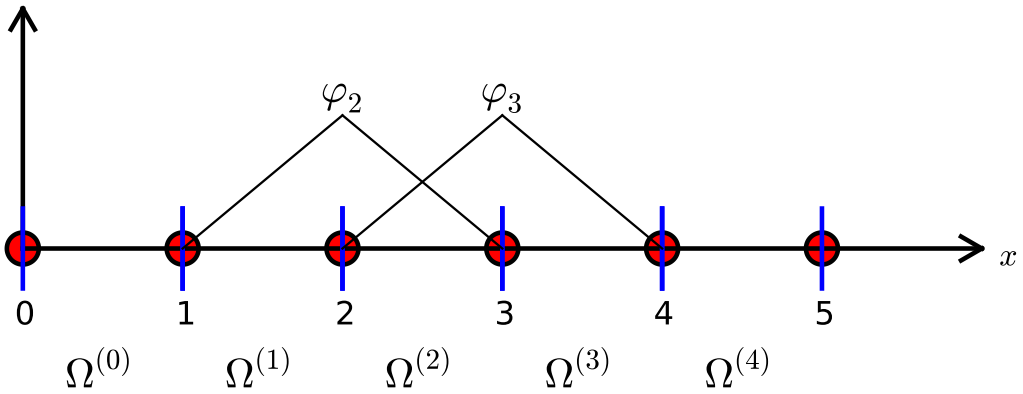
Funkcje bazowe o nośniku nieograniczonym: $\psi_i(x) \neq 0$ prawie w całym przedziale określoności

Nośnik funkcji: domknięcie zbioru argumentów funkcji, dla których ma ona wartość różną od zera (takie iksy dla których $f(x) \neq 0$)

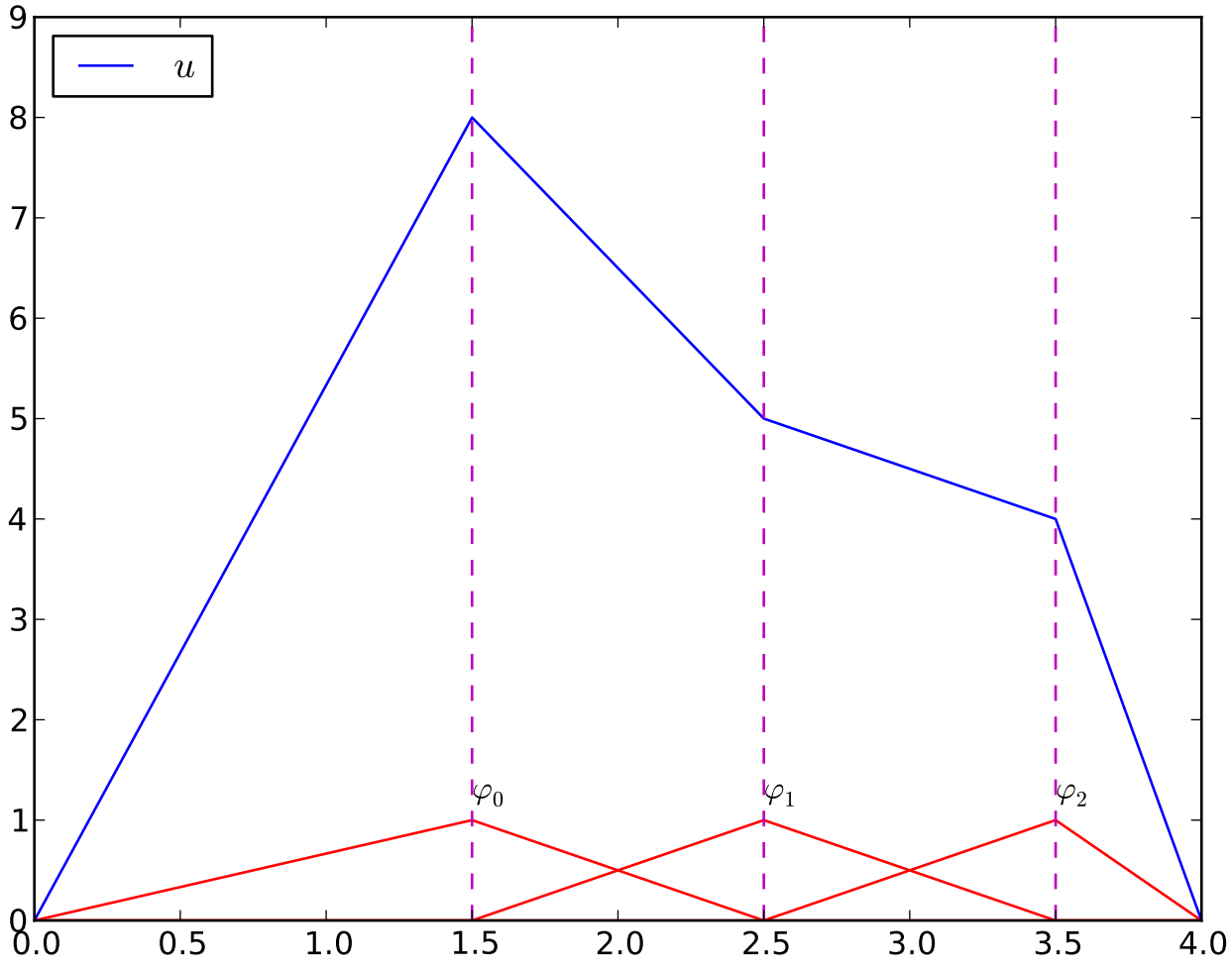


Funkcje bazowe o nośniku ograniczonym – FEM

- *Nośnik zwarty* (*Local support*): domknięcie zbioru tych x-ów, dla których $\psi_i(x) \neq 0$
- Typowe dla 1D - funkcje trójkątne (hat-shaped)
- $u(x)$ zbudowana przy pomocy takich funkcji ψ_i będzie funkcją przedziałami liniową
- Niech symbol φ_i oznacza odtąd tego typu funkcję trójkątną (przyjmijmy również $\psi_i = \varphi_i$)



Kombinacja liniowa funkcji trójkątnych



Elementy i węzły

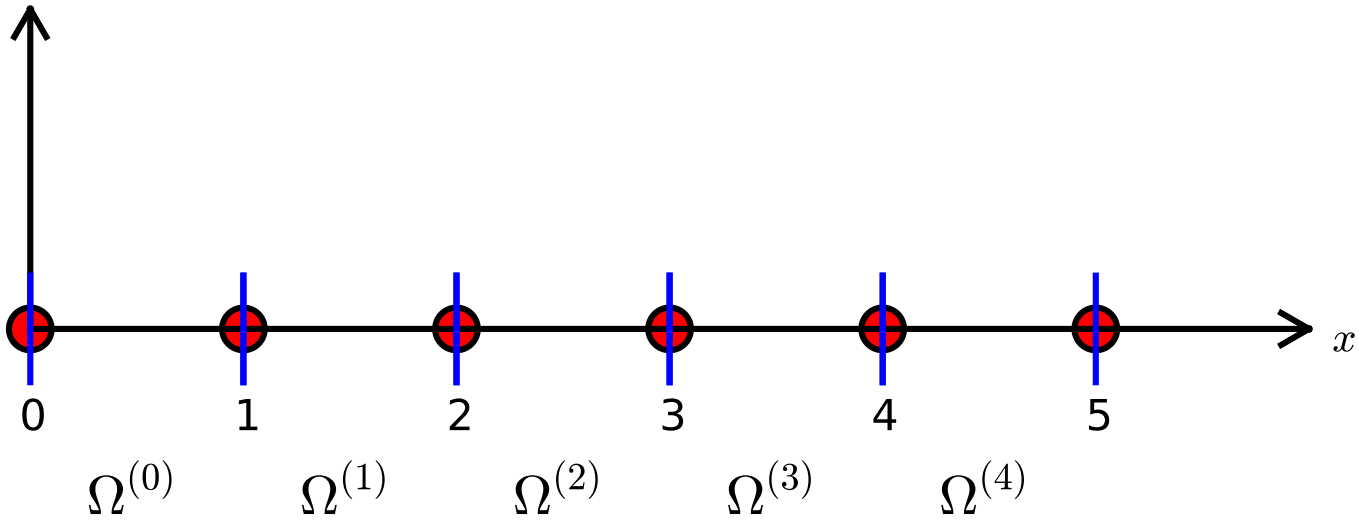
Podzielmy Ω na N_e rozdzielnych podobszarów podobszarów – *elementów*:

$$\Omega = \Omega^{(0)} \cup \ldots \cup \Omega^{(N_e)}$$

Na każdym elemencie wprowadzamy N_n *wezłów* (punktów): $x_0, \ldots, x_{N_n-1}$

- $\varphi_i(x)$ – i-ta funkcja bazowa
- $\varphi_i = 1$ w węźle i i $\varphi_i = 0$ w pozostałych węzłach
- φ_i to wielomian Lagrange'a na każdym elemencie
- Dla węzłów granicznych, leżących w punktach łączących dwa elementy funkcja φ_i jest zbudowane z wielomianów Lagrange'a na obu elementach

Przykład: obszar podzielony na elementy dwuwęzłowe (elementy typu P1)

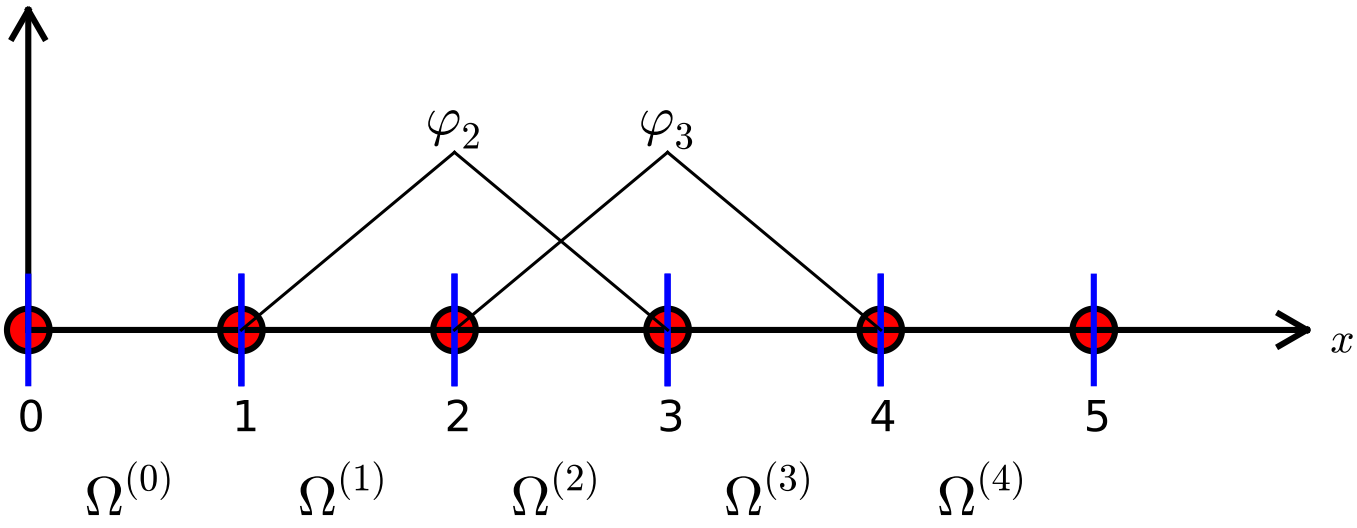


Struktura `nodes` – współrzędne węzłów.

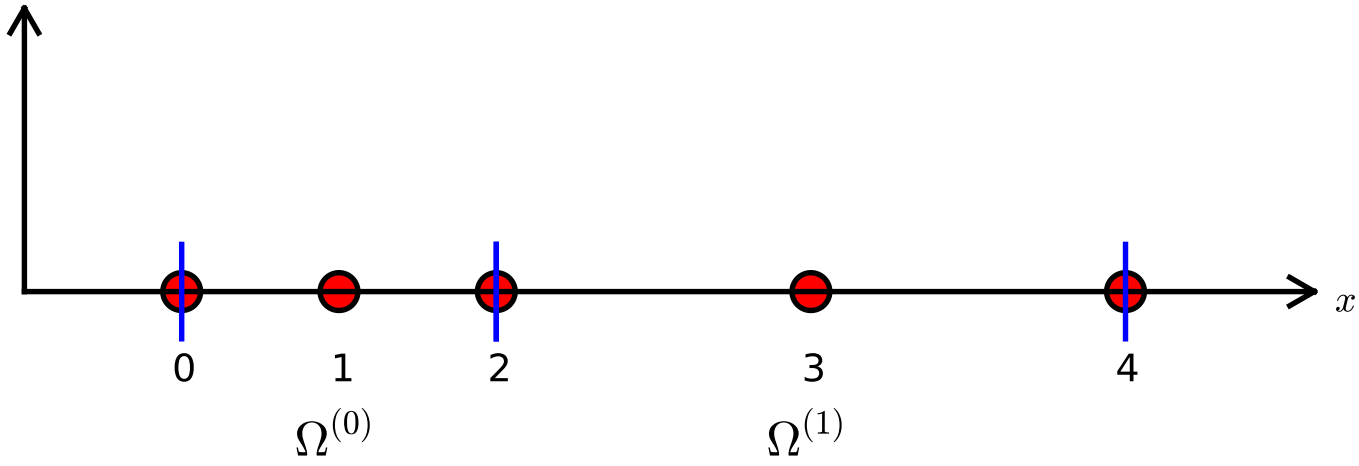
Struktura `elements` – numery (globalne) węzłów tworzących odpowiedni element.

```
nodes = [0, 1.2, 2.4, 3.6, 4.8, 5]
elements = [[0, 1], [1, 2], [2, 3], [3, 4], [4, 5]]
```

Przykład: dwie funkcje bazowe na siatce

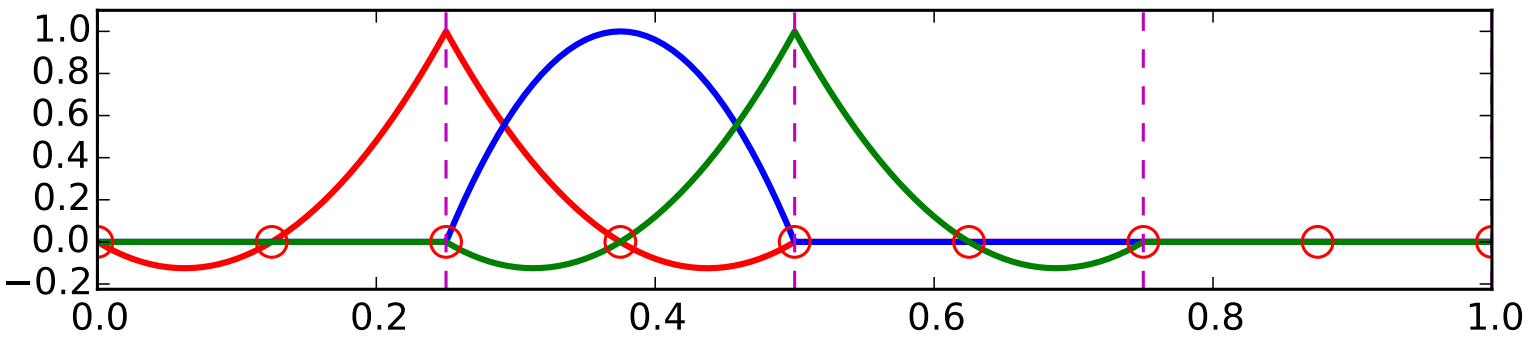


Przykład: elementy niejednorodne o trzech węzłach (elementy typu P2)

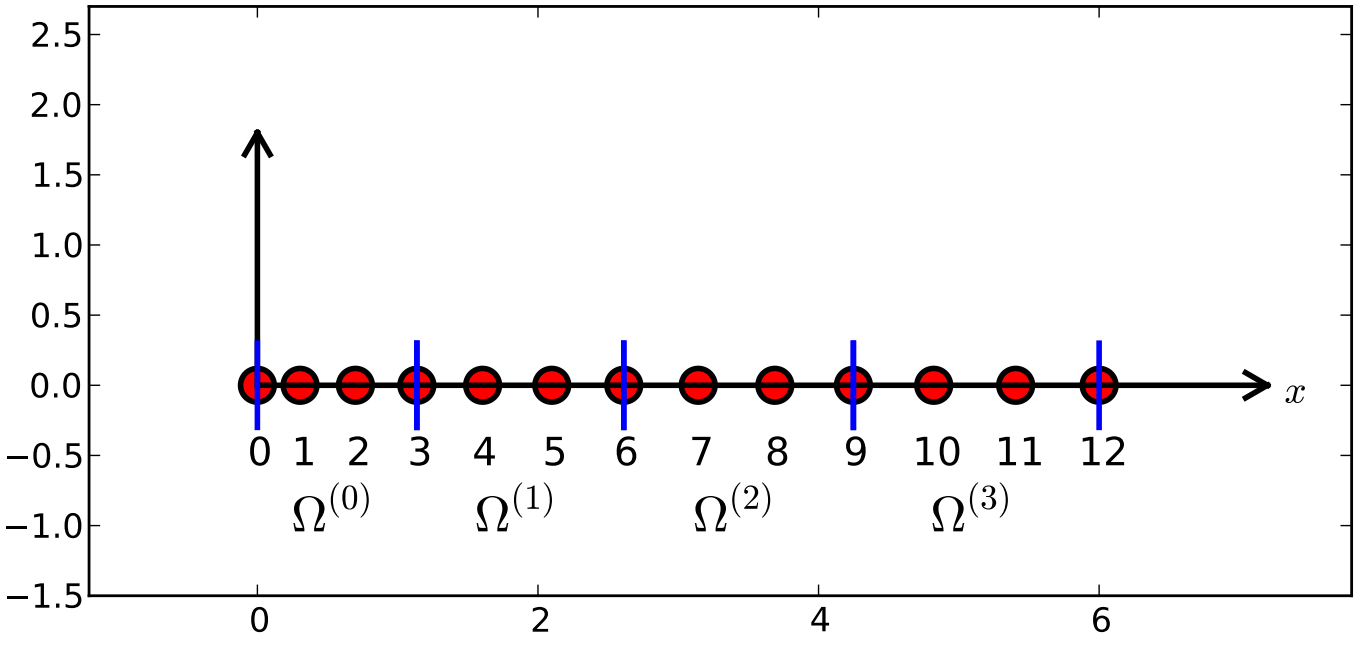



```
nodes = [0, 0.125, 0.25, 0.375, 0.5, 0.625, 0.75, 0.875, 1.0]
elements = [[0, 1, 2], [2, 3, 4], [4, 5, 6], [6, 7, 8]]
```

Przykład: funkcje bazowe na siatce (elementy typu P2)

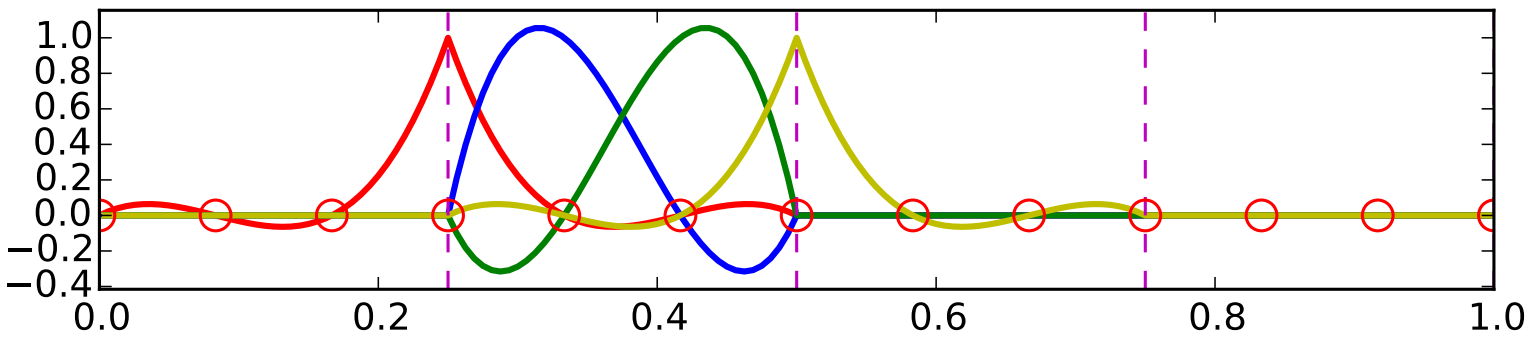


Przykład: elementy typu P3 (o czterech węzłach interpolacji)

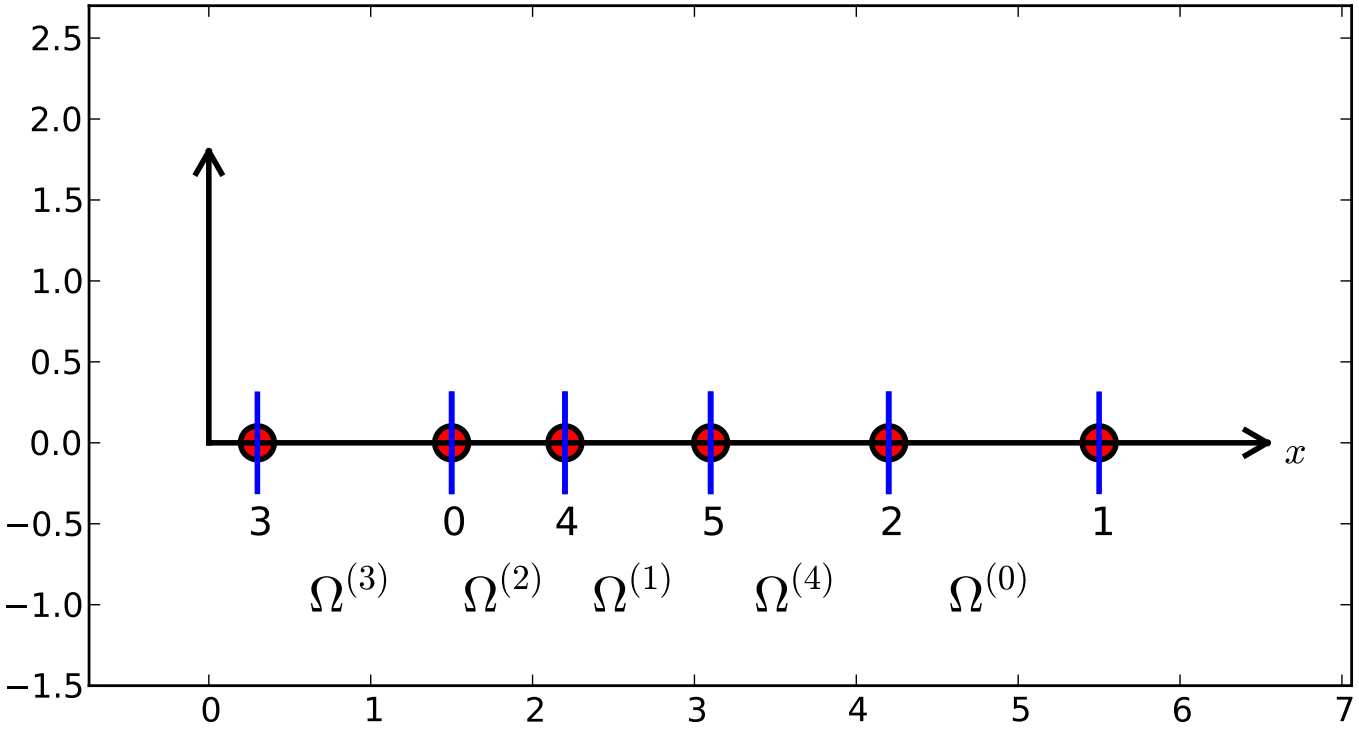



```
d = 3  # d+1 nodes per element
num_elements = 4
num_nodes = num_elements*d + 1
nodes = [i*0.5 for i in range(num_nodes)]
elements = [[i*d+j for j in range(d+1)] for i in range(num_elements)]
```

Przykład: funkcje bazowe na siatce (elementy typu P3)



Indeksacja nieregularna




```
nodes = [1.5, 5.5, 4.2, 0.3, 2.2, 3.1]
elements = [[2, 1], [4, 5], [0, 4], [3, 0], [5, 2]]
```

Współczynniki c_i – interpretacja

Ważna własność: c_i to wartość funkcji u w węźle i, x_i:

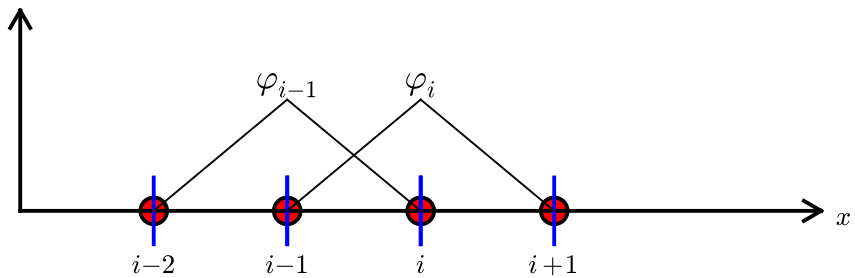
$$u(x_i) = \sum_{j \in \mathcal{I}_s} c_j \varphi_j(x_i) = c_i \varphi_i(x_i) = c_i$$

Powód: $\varphi_j(x_i) = 0$ jeśli $i \neq j$ i $\varphi_i(x_i) = 1$

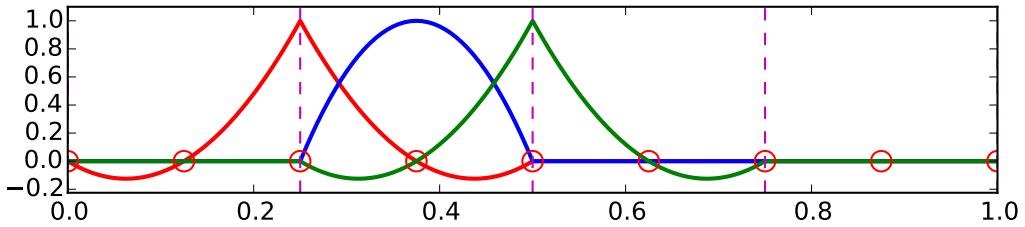
Własności funkcji bazowych

- $\varphi_i(x) \neq 0$ jedynie na tych elementach, które zawierają węzeł
  1. globalnym indeksie i
- $\varphi_i(x)\varphi_j(x) \neq 0$ wtedy i tylko wtedy gdy węzły i oraz j leżą na tym samym elemencie

Ponieważ $A_{i,j} = \int \varphi_i \varphi_j \, \mathrm{d}x$, *większość współczynników macierzy będzie równa zero* -> macierze rzadkie

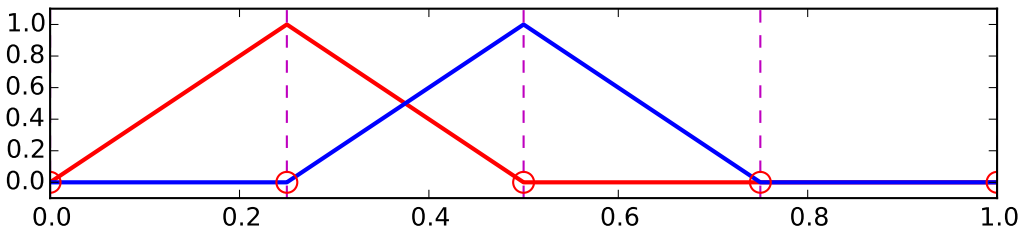


Konstrukcja kwadratowych φ_i (elementy typu P2)



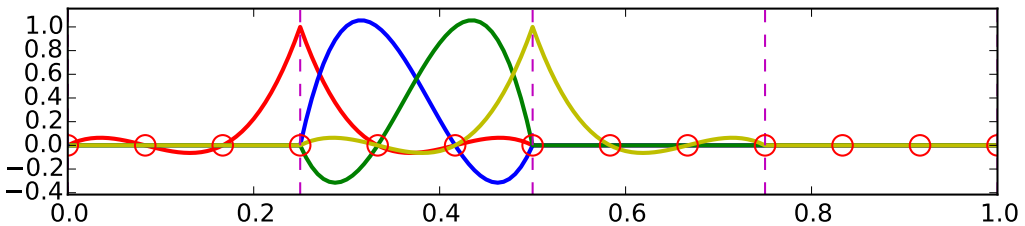
1. każdy węzeł elementu ma przypisany wielomian Lagrange'a
2. Wielomian o wartości 1 na brzegu elementu należy "połączyć" z wielomianem z sąsiedniego elementu, który ma wartość 1 w tym samym punkcie

Liniowe φ_i (elementy typu P1)

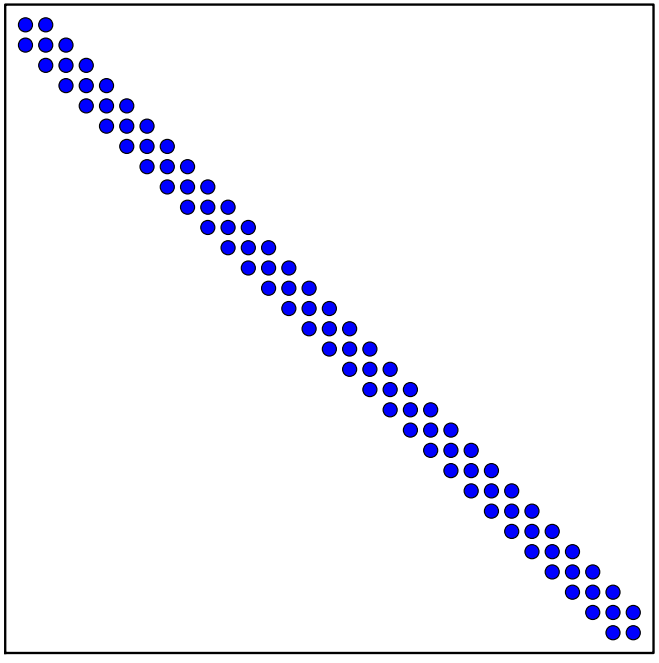


$$\varphi_i(x) = \begin{cases} 0, & x < x_{i-1} \\ (x - x_{i-1})/h & x_{i-1} \leq x < x_i \\ 1 - (x - x_i)/h, & x_i \leq x < x_{i+1} \\ 0, & x \geq x_{i+1} \end{cases}$$

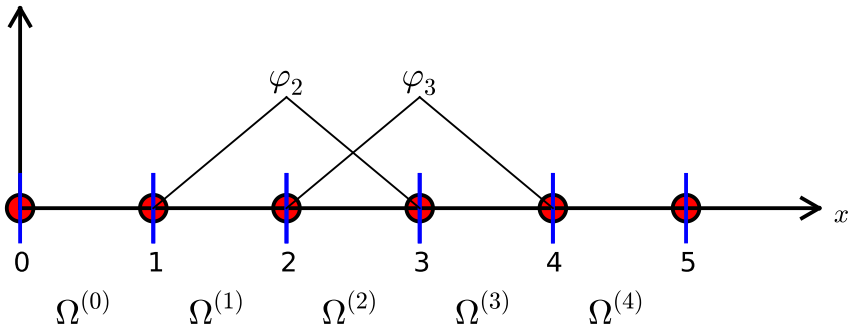
Sześcienne φ_i (elementy typu P3)



Generowanie URL



Przykład 1: Obliczenie wartości (niediagonalnego) elementu macierzy



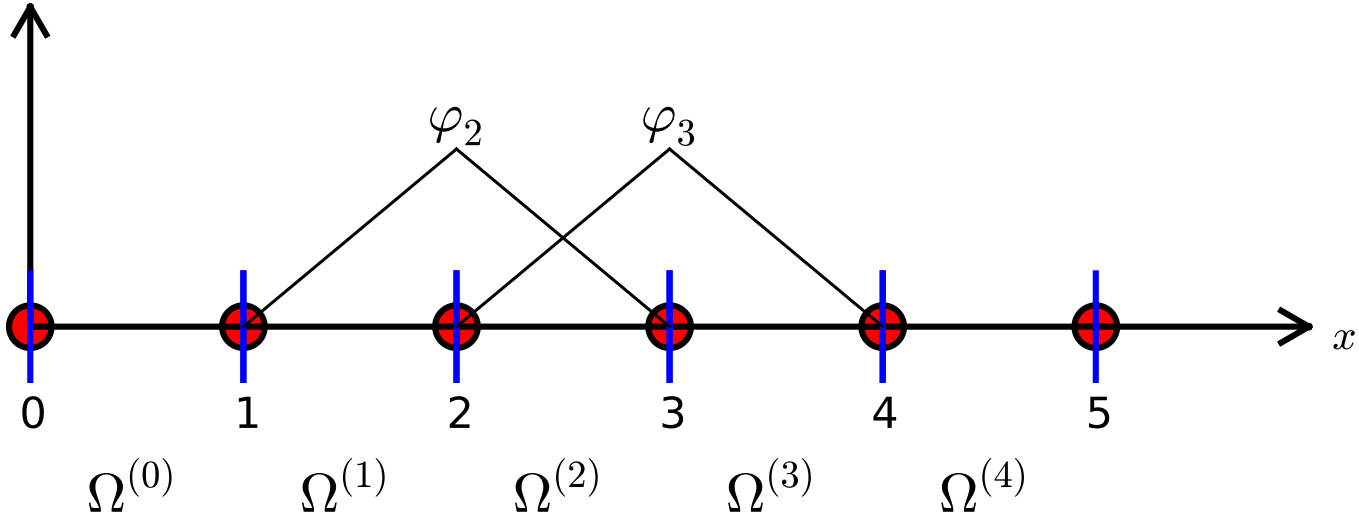
Uproszczenie: elementy jednakowej długości.

$A_{2,3} = \int_\Omega \varphi_2 \varphi_3 dx$: $\varphi_2 \varphi_3 \neq 0$ jedynie na elemencie 2. Dla tego elementu:

$$\varphi_3(x) = (x - x_2)/h, \quad \varphi_2(x) = 1 - (x - x_2)/h$$

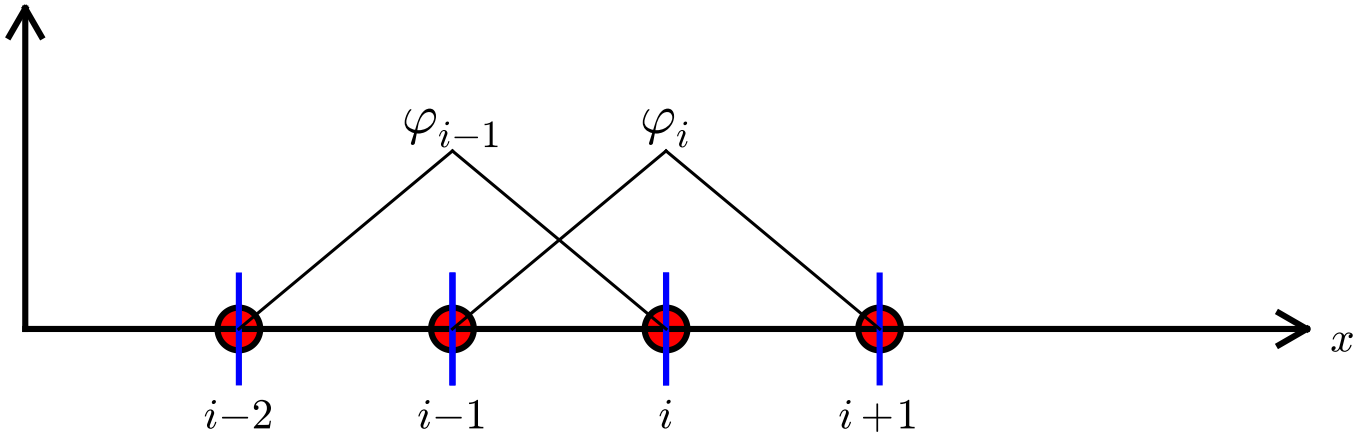
$$A_{2,3} = \int_\Omega \varphi_2 \varphi_3 \, \mathrm{d}x = \int_{x_2}^{x_3} \left(1 - \frac{x - x_2}{h}\right) \frac{x - x_2}{h} \, \mathrm{d}x = \frac{h}{6}$$

Przykład 2: Obliczenie wartości (diagonalnego) elementu macierzy



$$A_{2,2} = \int_{x_1}^{x_2} \left(\frac{x - x_1}{h} \right)^2 \mathrm{d}x + \int_{x_2}^{x_3} \left(1 - \frac{x - x_2}{h} \right)^2 \mathrm{d}x = \frac{2h}{3}$$

Ogólna postać wzoru na wartość elementu A_{ij} - rysunek



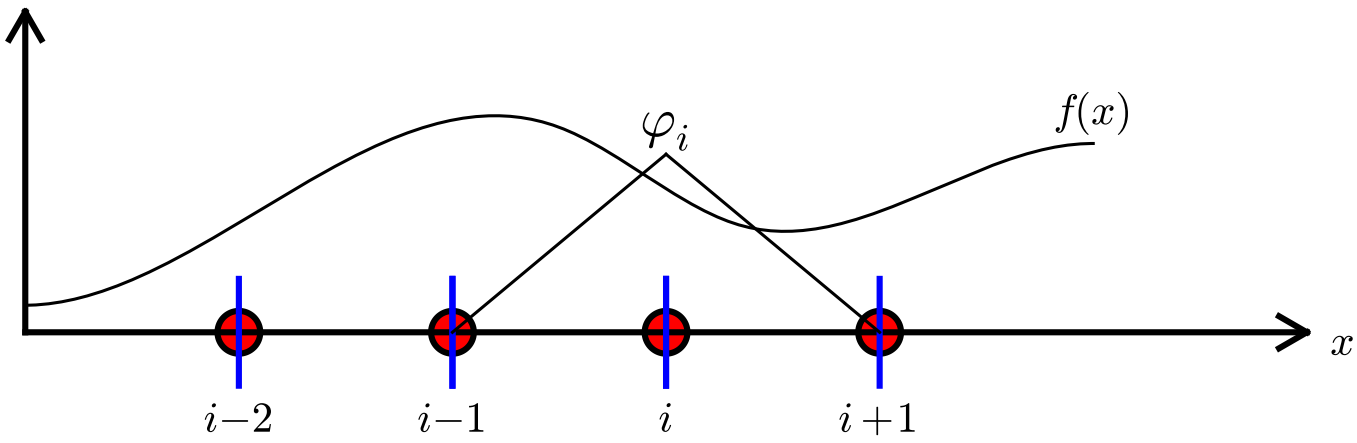
$$A_{i,i-1} = \int_\Omega \varphi_i \varphi_{i-1} \, \mathrm{d}x = ?$$

Ogólna postać wzoru na wartość elementu A_{ij} - obliczenia

$$
\begin{aligned}
A_{i,i-1} &= \int_{\Omega} \varphi_i \varphi_{i-1} \, \mathrm{d}x \\
&= \underbrace{\int_{x_{i-2}}^{x_{i-1}} \varphi_i \varphi_{i-1} \, \mathrm{d}x}_{\varphi_i = 0} + \int_{x_{i-1}}^{x_i} \varphi_i \varphi_{i-1} \, \mathrm{d}x + \underbrace{\int_{x_i}^{x_{i+1}} \varphi_i \varphi_{i-1} \, \mathrm{d}x}_{\varphi_{i-1} = 0} \\
&= \int_{x_{i-1}}^{x_i} \underbrace{\left(\frac{x - x_i}{h} \right)}_{\varphi_i(x)} \underbrace{\left(1 - \frac{x - x_{i-1}}{h} \right)}_{\varphi_{i-1}(x)} \mathrm{d}x = \frac{h}{6}
\end{aligned}
$$

- $A_{i,i+1} = A_{i,i-1}$ ze względu na symetrię
- $A_{i,i} = 2h/3$ (obliczenia jak w przypadku $A_{2,2}$), z wyjątkiem:
- $A_{0,0} = A_{N,N} = h/3$ (całka tylko na jednym elemencie)

Obliczenia dla prawej strony równania



$$b_i = \int_\Omega \varphi_i(x) f(x)\,\mathrm{d}x = \int_{x_{i-1}}^{x_i} \frac{x - x_{i-1}}{h} f(x)\,\mathrm{d}x + \int_{x_i}^{x_{i+1}} \left(1 - \frac{x - x_i}{h}\right) f(x)$$

Do dalszych obliczeń potrzebna konkretna postać $f(x)$ ...

Przykład: rozwiązanie dla obszaru dwu-elementowego – URL i rozwiązanie

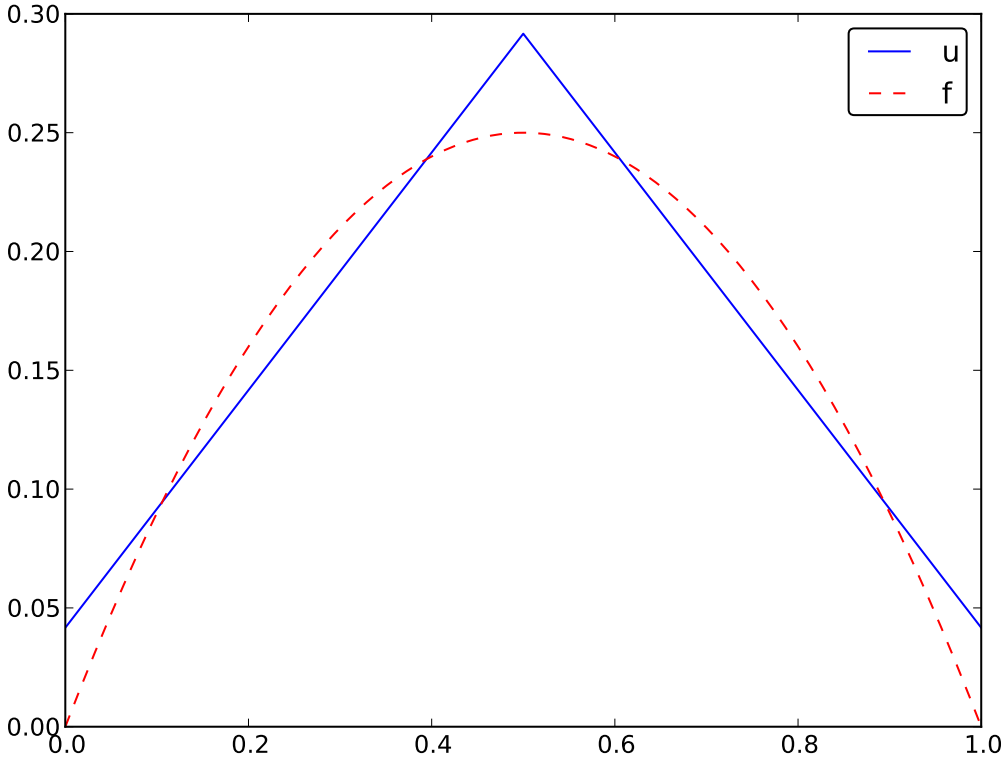
- $f(x) = x(1-x)$ na $\Omega = [0,1]$
- Dwa elementy o jednakowej długości: $[0, 0.5]$ oraz $[0.5, 1]$

$$A = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 \\ 1 & 4 & 1 \\ 0 & 1 & 2 \end{pmatrix}, \quad b = \frac{h^2}{12}\begin{pmatrix} 2-3h \\ 12-14h \\ 10-17h \end{pmatrix}$$

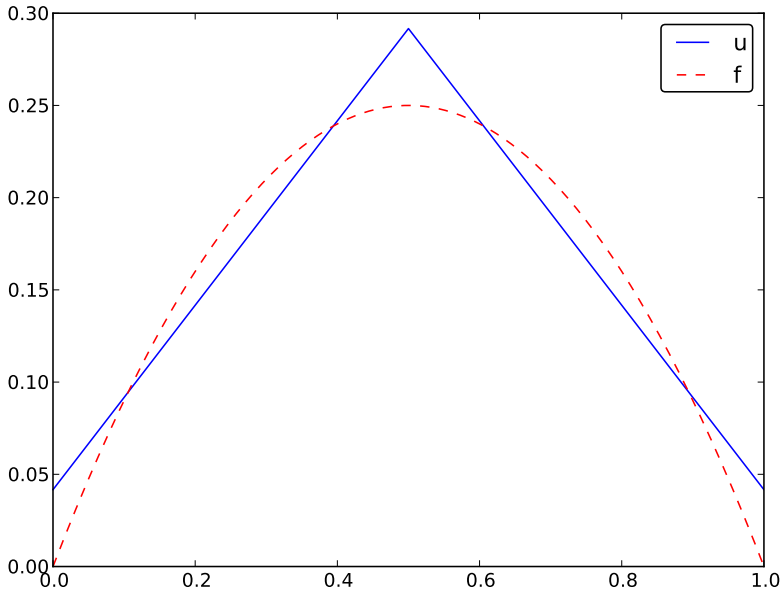
$$c_0 = \frac{h^2}{6}, \quad c_1 = h - \frac{5}{6}h^2, \quad c_2 = 2h - \frac{23}{6}h^2$$

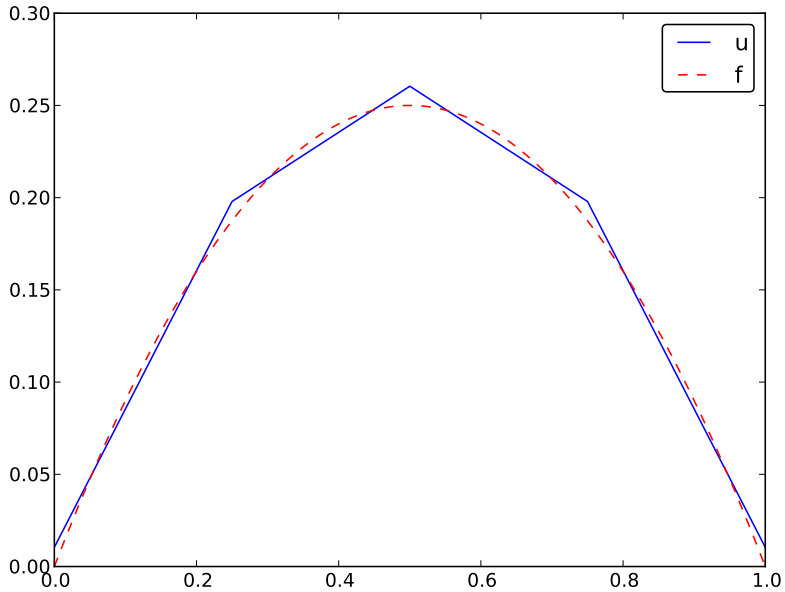
Przykład: rozwiązanie dla obszaru dwu-elementowego – rozwiązanie-rysunek

$$u(x) = c_0\varphi_0(x) + c_1\varphi_1(x) + c_2\varphi_2(x)$$



Przykład: Rozwiązanie dla obszaru 4-elementowego – rozwiązanie-rysunek





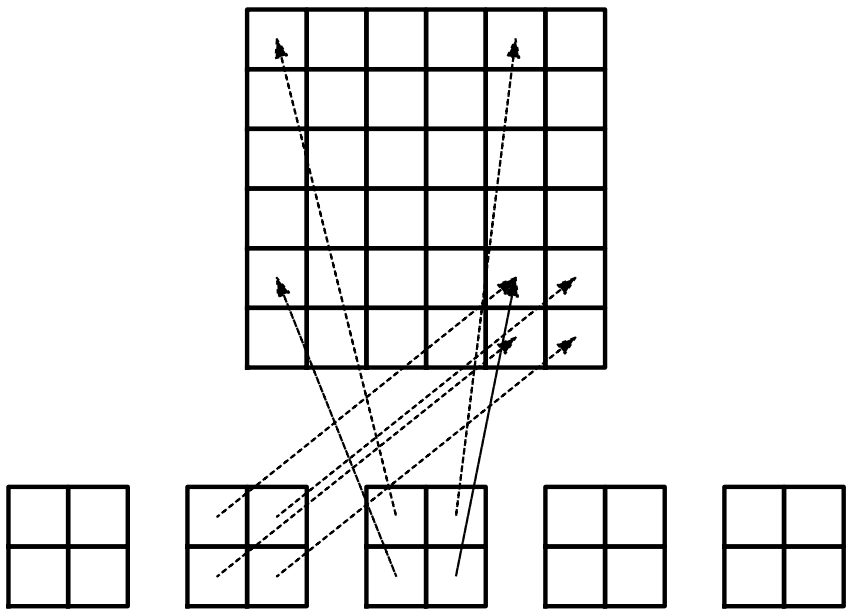
Przykład: elementy typu P2

Przypomnienie: jeśli $f \in V$, u odtworzy rozwiązanie bezbłędnie. Jeśli f to parabola, dowolna siatka elementów typu P2 (1 lub wiele elementów) sprawi wygeneruje $u = f$. To samo dotyczyć będzie elementów typu P3, P4, itd., ponieważ one wszystkie potrafią odtworzyć wielomian 2. stopnia bezbłędnie.

Generowanie macierzy globalnej - logika obliczeń

(ang. assemble - gromadzić, składać, zbierać)

assembling, assemblacja?

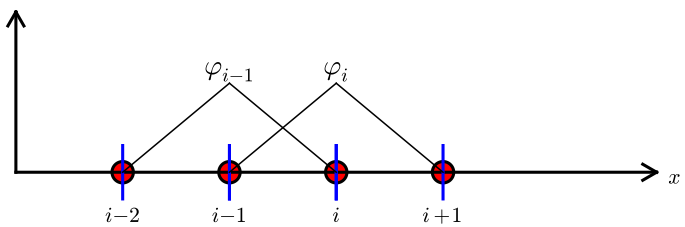


Całkowanie z perspektywy elementu

$$A_{i,j} = \int_\Omega \varphi_i \varphi_j dx = \sum_e \int_{\Omega^{(e)}} \varphi_i \varphi_j dx, \quad A_{i,j}^{(e)} = \int_{\Omega^{(e)}} \varphi_i \varphi_j dx$$

Ważne spostrzeżenia:

- $A_{i,j}^{(e)} \neq 0$ wtedy i tylko wtedy gdy węzeły i oraz j leżą na tym samym elemencie e (w przeciwnym wypadku nośniki funkcji to zbiory rozłączne)
- Wszystkie niezerowe współczynniki danego elementu $A_{i,j}^{(e)}$ tworzą lokalną macierz dla danego elementu (*element matrix*)
- "Wkład" w macierz lokalną elementu mają wyłącznie funkcje bazowe związane z węzłami leżącymi na tym elemencie
- Wygodne rozwiązanie: wprowadzenie *indeksacji lokalnej* węzłów leżących na danym elemencie: $0, 1, \ldots, d$



Macierz elementów: indeksacja lokalna/indeksacja globalna

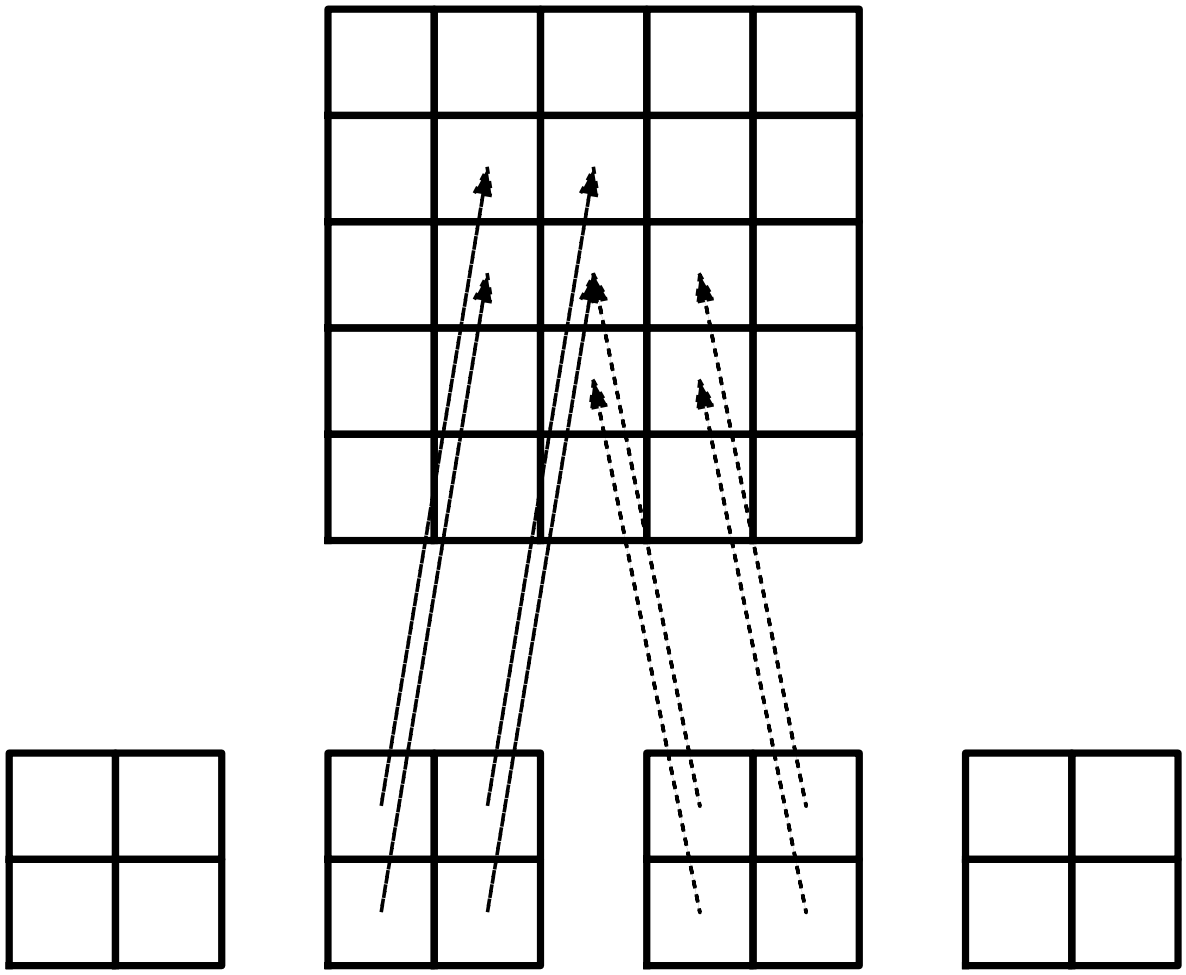
$$\tilde{A}^{(e)} = \{\tilde{A}^{(e)}_{r,s}\}, \quad \tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)} \varphi_{q(e,s)} dx, \quad r,s \in I_d = \{0,\ldots,d\}$$

- *r, s lokalne indeksy węzłów* na elemencie: $0, 1, \ldots, d$
- *i, j globalne indeksy węzłów* $i, j \in \mathcal{I}_s = \{0, 1, \ldots, N\}$
- $i = q(e, r)$: transformacja lokalnej indeksacji w globalną (matematyczny zapis pythonowskiego `i=elements[e][r]`) gdzie: `elements = [[1, 2],[2,3],[3,4], ..., [7,8],[8,9,10], ...]`
- Uwzględnienie macierzy lokalnej $\tilde{A}^{(e)}_{r,s}$ w macierzy globalnej $A_{i,j}$ (*assembly*)

$$A_{q(e,r),q(e,s)} := A_{q(e,r),q(e,s)} + \tilde{A}^{(e)}_{r,s}, \quad r,s \in I_d$$

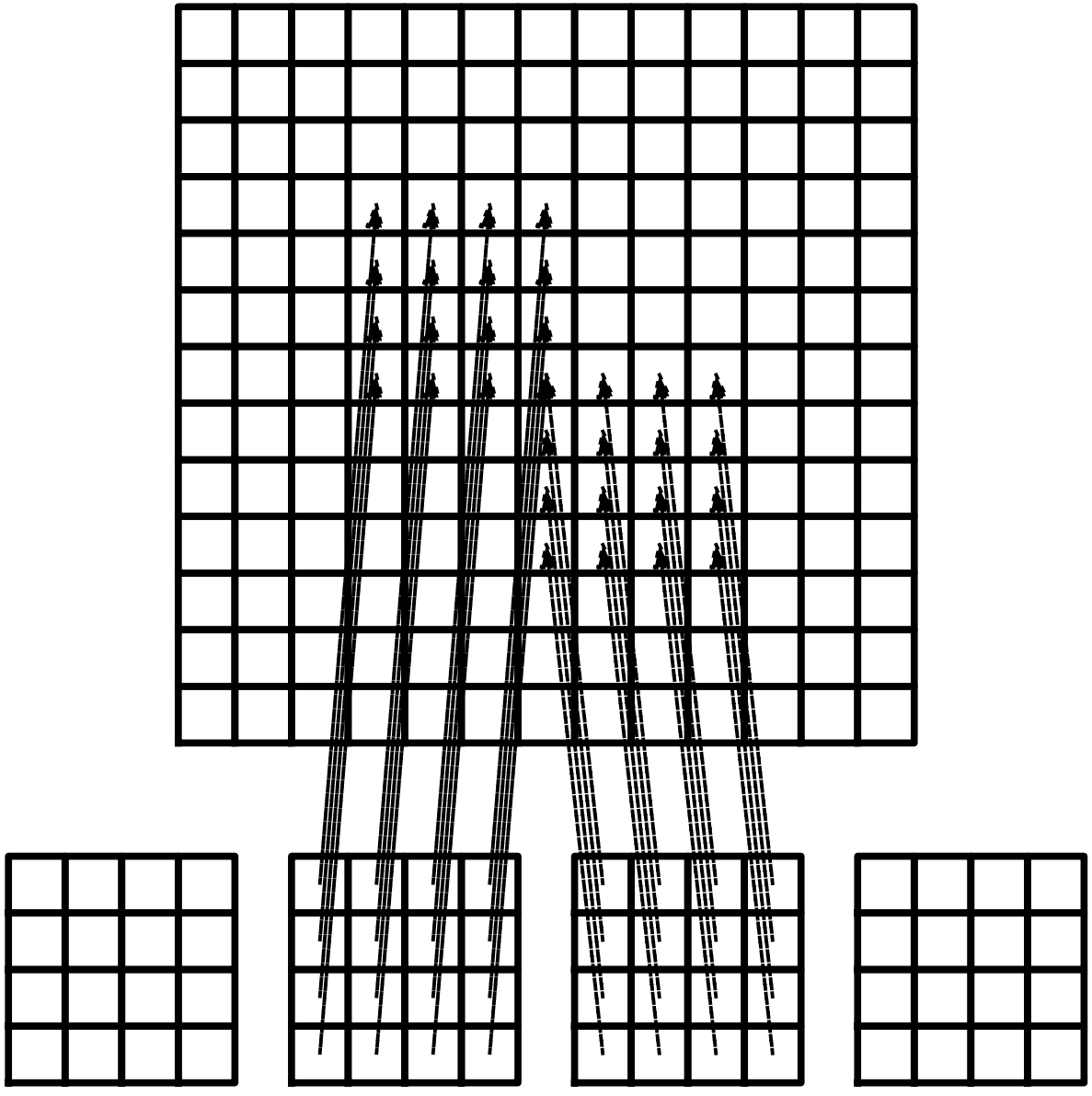
i	e	r
1	1	1
2	1	2
2	2	1
3	2	2
⋮	⋮	⋮
8	7	2
8	8	1
9	8	2
10	8	3
10	9	1
⋮	⋮	⋮

Przykład: assembling macierzy dla kolejno ponumerowanych elementów P1

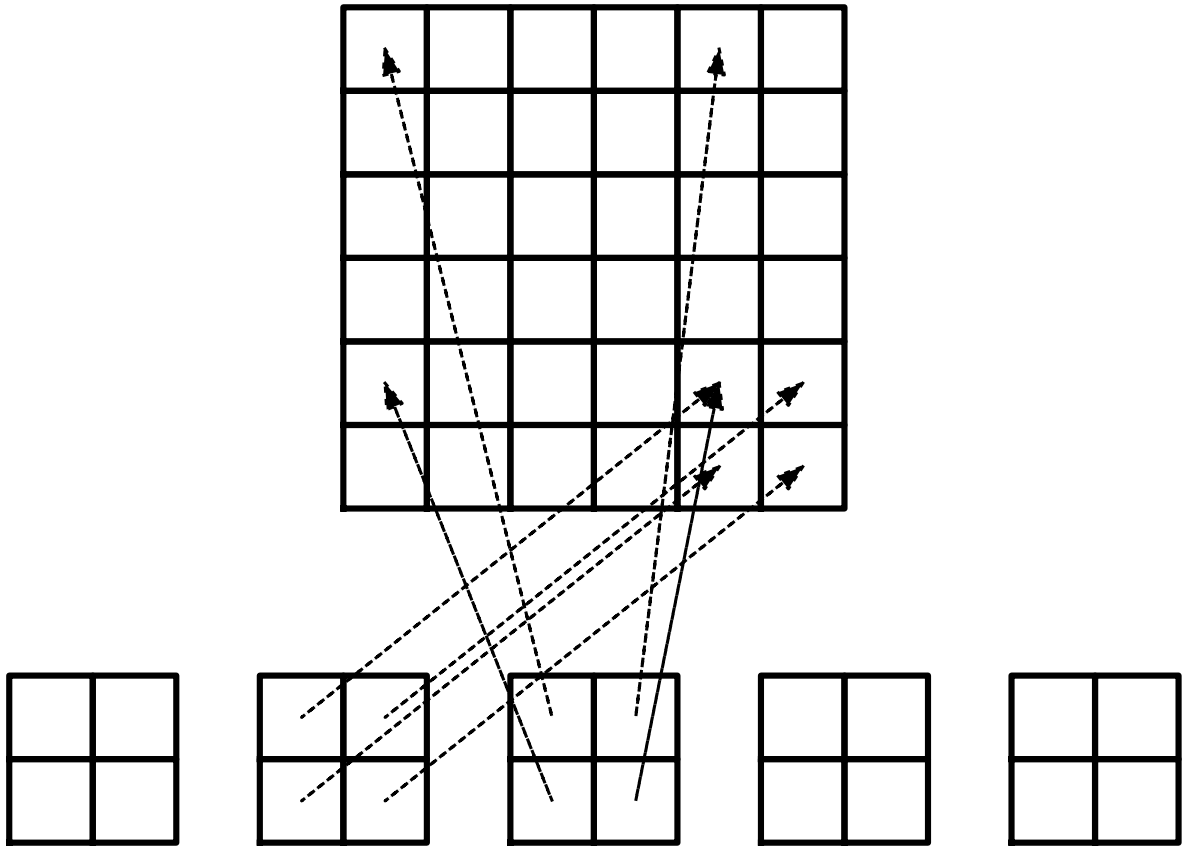


TODO

Przykład: assembling macierzy dla kolejno ponumerowanych elementów P3



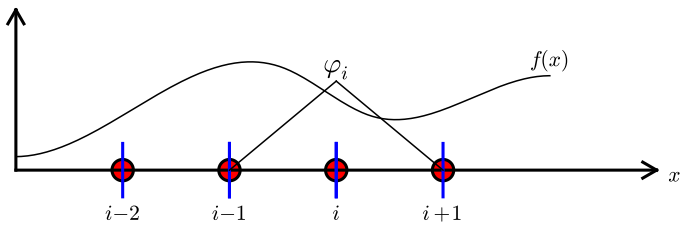
Przykład: assembling macierzy dla nieregularnej siatki elementów P1



TODO

Assembling prawej strony układu

$$b_i = \int_\Omega f(x)\varphi_i(x)dx = \sum_e \int_{\Omega^{(e)}} f(x)\varphi_i(x)dx, \quad b_i^{(e)} = \int_{\Omega^{(e)}} f(x)\varphi_i(x)dx$$



Ważne spostrzeżenia:

- $b_i^{(e)} \neq 0$ wtedy i tylko wtedy gdy węzeł globalny i leży na danym elemencie e (w przeciwnym przypadku $\varphi_i = 0$)
- $d+1$ niezerowych wartości $b_i^{(e)}$ może być zgromadzonych w *lokalnym wektorze* elementu e. $\tilde{b}_r^{(e)} = \{\tilde{b}_r^{(e)}\}$, $r \in I_d$

Assembling:

$$b_{q(e,r)} := b_{q(e,r)} + \tilde{b}_r^{(e)}, \quad r \in I_d$$

Transformacja współrzędnych globalnych do współrzędnych unormowanych

Normalizacja współrzędnych położenia:

Zamiast całkować w granicach $[x_L, x_R]$

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx = \int_{x_L}^{x_R} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx$$

można transformować przedział $[x_L, x_R]$ na przedział unormowany $[-1, 1]$ o współrzędnej lokalnej X

Transformacja liniowa $X \in [-1, 1]$ w $x \in [x_L, x_R]$

(Transformacja afiniczna)

$$x = \frac{1}{2}(x_L + x_R) + \frac{1}{2}(x_R - x_L)X$$

inaczej

$$x = x_m + \frac{1}{2}hX, \qquad x_m = (x_L + x_R)/2, \quad h = x_R - x_L$$

Transformacja odwrotna:

$$X = \frac{2x + (x_L + x_R)}{(x_R - x_L)}$$

Transformacja całki

Zmiana granic całkowania -> całkowanie na przedziale unormowanym: podstawienie $x(X)$ w miejsce x.

Lokalne funkcje bazowe we współrzędnych unormowanych:

$$\tilde{\varphi}_r(X) = \varphi_{q(e,r)}(x(X))$$

$$x = \tfrac{1}{2}(x_L + x_R) + \tfrac{1}{2}(x_R - x_L)X$$
$$\downarrow$$
$$dx = \tfrac{1}{2}(x_R - x_L)dX$$

$$\tilde{A}^{(e)}_{r,s} = \int_{\Omega^{(e)}} \varphi_{q(e,r)}(x)\varphi_{q(e,s)}(x)dx = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X) \underbrace{\frac{dx}{dX}}_{\det J = h/2} dX$$

$$\tilde{A}^{(e)}_{r,s} = \int_{-1}^{1} \tilde{\varphi}_r(X)\tilde{\varphi}_s(X) \det J \, dX$$

$$\tilde{b}^{(e)}_{r} = \int_{\Omega^{(e)}} f(x)\varphi_{q(e,r)}(x)dx = \int_{-1}^{1} f(x(X))\tilde{\varphi}_r(X) \det J \, dX$$

Zalety całkowania na przedziale unormowanym

- Całkowanie zawsze w tych samych granicach całkowania $[-1, 1]$
- Potrzebne wzory tylko dla $\tilde{\varphi}_r(X)$ na jednym elemencie (brak funkcji definiowanych na przedziałach (piecewise polynomial))
- Funkcja $\tilde{\varphi}_r(X)$ jest taka sama dla wszystkich elementów niezależnie od ich położenia i rozmiarów (długości). Długość odcinka jest uwzględniona poprzez jakobian $\det J$

Funkcje bazowe P1 na elemencie unormowanym

$$\tilde{\varphi}_0(X) = \frac{1}{2}(1 - X) \tag{8}$$

$$\tilde{\varphi}_1(X) = \frac{1}{2}(1 + X) \tag{9}$$

(proste funkcje wielomianowe zamiast definicji funkcji na podprzedziałach

Funkcje bazowe P2 na elemencie unormowanym

$$\tilde{\varphi}_0(X) = \frac{1}{2}(X-1)X \tag{10}$$

$$\tilde{\varphi}_1(X) = 1 - X^2 \tag{11}$$

$$\tilde{\varphi}_2(X) = \frac{1}{2}(X+1)X \tag{12}$$

Łatwość wygenerowania elementów dowolnego rzędu... Jak?

Sposoby znalezienia wzorów na funkcje bazowe

1. Transformacja globalnych funkcji bazowych $\varphi_i(x)$ na element unormowany ze współrzędną X
2. Obliczenie $\tilde{\varphi}_r(X)$
   - dla zadanego stopnia d szukamy wielomianów opartych o węzły wewnątrz przedziału $[-1, 1]$ o własności
     - $\tilde{\varphi}_r(X) = 1$ w węźle r
     - $\tilde{\varphi}_r(X) = 0$ we wszystkich pozostałych d węzłach
3. Wykorzystanie wzoru interpolacyjnego Lagrange'a

Całkowanie po elemencie unormowanym - lokalna macierz

Założenie: elementy typu P1, oraz funkcja $f(x) = x(1-x)$.

$$\tilde{A}^{(e)}_{0,0} = \int_{-1}^{1} \tilde{\varphi}_0(X)\tilde{\varphi}_0(X)\frac{h}{2}dX$$

$$= \int_{-1}^{1} \frac{1}{2}(1-X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X)^2 dX = \frac{h}{3} \quad (13)$$

$$\tilde{A}^{(e)}_{1,0} = \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_0(X)\frac{h}{2}dX$$

$$= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1-X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1-X^2) dX = \frac{h}{6} \quad (14)$$

$$\tilde{A}^{(e)}_{0,1} = \tilde{A}^{(e)}_{1,0} \quad (15)$$

$$\tilde{A}^{(e)}_{1,1} = \int_{-1}^{1} \tilde{\varphi}_1(X)\tilde{\varphi}_1(X)\frac{h}{2}dX$$

$$= \int_{-1}^{1} \frac{1}{2}(1+X)\frac{1}{2}(1+X)\frac{h}{2}dX = \frac{h}{8}\int_{-1}^{1}(1+X)^2 dX = \frac{h}{3} \quad (16)$$

Całkowanie po elemencie unormowanym - wektor prawej strony

$$\tilde{b}_0^{(e)} = \int_{-1}^{1} f(x(X))\tilde{\varphi}_0(X)\frac{h}{2}dX$$
$$= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1 - (x_m + \frac{1}{2}hX))\frac{1}{2}(1 - X)\frac{h}{2}dX$$
$$= -\frac{1}{24}h^3 + \frac{1}{6}h^2 x_m - \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m \quad (17)$$
$$\tilde{b}_1^{(e)} = \int_{-1}^{1} f(x(X))\tilde{\varphi}_1(X)\frac{h}{2}dX$$
$$= \int_{-1}^{1} (x_m + \frac{1}{2}hX)(1 - (x_m + \frac{1}{2}hX))\frac{1}{2}(1 + X)\frac{h}{2}dX$$
$$= -\frac{1}{24}h^3 - \frac{1}{6}h^2 x_m + \frac{1}{12}h^2 - \frac{1}{2}hx_m^2 + \frac{1}{2}hx_m \quad (18)$$

x_m: środek elementu

Obliczenia symboliczne zamiast żmudnego liczenia na kartce...

```
>>> import sympy as sym
>>> x, x_m, h, X = sym.symbols('x x_m h X')
>>> sym.integrate(h/8*(1-X)**2, (X, -1, 1))
h/3
>>> sym.integrate(h/8*(1+X)*(1-X), (X, -1, 1))
h/6
>>> x = x_m + h/2*X
>>> b_0 = sym.integrate(h/4*x*(1-x)*(1-X), (X, -1, 1))
>>> print b_0
-h**3/24 + h**2*x_m/6 - h**2/12 - h*x_m**2/2 + h*x_m/2
```

Implementacja

- Funkcje przedstawione na kolejnych slajdach znajdują sie w module `fe_approx1D.py`
- Przedstawione funkcje działają w trybie symbolicznym, jak i numerycznym
- Kod zawiera wszystkie kroku obliczeń elementami skończonymi.

Generowanie funkcji bazowych na przedziale unormowanymi

Niech $\tilde{\varphi}_r(X)$ będzie wielomianem Lagrange'a stopnia d:

```
import sympy as sym
import numpy as np

def phi_r(r, X, d):
    if isinstance(X, sym.Symbol):
        h = sym.Rational(1, d)  # node spacing
        nodes = [2*i*h - 1 for i in range(d+1)]
    else:
        # assume X is numeric: use floats for nodes
        nodes = np.linspace(-1, 1, d+1)
    return Lagrange_polynomial(X, r, nodes)

def Lagrange_polynomial(x, i, points):
    p = 1
    for k in range(len(points)):
        if k != i:
            p *= (x - points[k])/(points[i] - points[k])
    return p

def basis(d=1):
    """Return the complete basis."""
    X = sym.Symbol('X')
    phi = [phi_r(r, X, d) for r in range(d+1)]
    return phi
```

Obliczanie współczynników macierzy

```
def element_matrix(phi, Omega_e, symbolic=True):
    n = len(phi)
    A_e = sym.zeros((n, n))
    X = sym.Symbol('X')
    if symbolic:
        h = sym.Symbol('h')
    else:
        h = Omega_e[1] - Omega_e[0]
    detJ = h/2  # dx/dX
    for r in range(n):
        for s in range(r, n):
            A_e[r,s] = sym.integrate(phi[r]*phi[s]*detJ, (X, -1, 1))
            A_e[s,r] = A_e[r,s]
    return A_e
```

Przykład: Obliczenia macierzy współczynników: symbolicznie vs numerycznie

```
>>> from fe_approx1D import *
>>> phi = basis(d=1)
>>> phi
[1/2 - X/2, 1/2 + X/2]
>>> element_matrix(phi, Omega_e=[0.1, 0.2], symbolic=True)
[h/3, h/6]
[h/6, h/3]
>>> element_matrix(phi, Omega_e=[0.1, 0.2], symbolic=False)
[0.033333333333333, 0.016666666666667]
[0.016666666666667, 0.033333333333333]
```

Obliczenia współczynników wektora prawej strony

```
def element_vector(f, phi, Omega_e, symbolic=True):
    n = len(phi)
    b_e = sym.zeros((n, 1))
    # Make f a function of X
    X = sym.Symbol('X')
    if symbolic:
        h = sym.Symbol('h')
    else:
        h = Omega_e[1] - Omega_e[0]
    x = (Omega_e[0] + Omega_e[1])/2 + h/2*X  # mapping
    f = f.subs('x', x)  # substitute mapping formula for x
    detJ = h/2  # dx/dX
    for r in range(n):
        b_e[r] = sym.integrate(f*phi[r]*detJ, (X, -1, 1))
    return b_e
```

Zwróć uwagę na `f.subs('x', x)` -> podstawienie $x(X)$ za `x` w formule na `f` (od teraz `f` jest funkcją $f(X)$)

Powrót do całkowania numerycznego w razie niepowodzenia całkowania symbolicznego $\int f\tilde{\varphi}_r dx$

- Macierz lewej strony: tylko wielomiany -> sympy zawsze da radę
- Wektor prawej strony: całkowanie $\int f\tilde{\varphi}\,\mathrm{d}x$ może się nie powieść (sympy zwróci obiekt typu `Integral` zamiast liczby)

```
def element_vector(f, phi, Omega_e, symbolic=True):
        ...
        I = sym.integrate(f*phi[r]*detJ, (X, -1, 1))  # try...
        if isinstance(I, sym.Integral):
            h = Omega_e[1] - Omega_e[0]  # Ensure h is numerical
            detJ = h/2
            integrand = sym.lambdify([X], f*phi[r]*detJ)
            I = sym.mpmath.quad(integrand, [-1, 1])
        b_e[r] = I
        ...
```

Assembling URL i rozwiązanie

```
def assemble(nodes, elements, phi, f, symbolic=True):
    N_n, N_e = len(nodes), len(elements)
    zeros = sym.zeros if symbolic else np.zeros
    A = zeros((N_n, N_n))
    b = zeros((N_n, 1))
    for e in range(N_e):
        Omega_e = [nodes[elements[e][0]], nodes[elements[e][-1]]]

        A_e = element_matrix(phi, Omega_e, symbolic)
        b_e = element_vector(f, phi, Omega_e, symbolic)

        for r in range(len(elements[e])):
            for s in range(len(elements[e])):
                A[elements[e][r],elements[e][s]] += A_e[r,s]
            b[elements[e][r]] += b_e[r]
    return A, b
```

Rozwiązanie URL

```
if symbolic:
    c = A.LUsolve(b)            # sympy arrays, symbolic Gaussian elim.
else:
    c = np.linalg.solve(A, b)   # numpy arrays, numerical solve
```

Uwaga: obliczanie współczynników macierzy `A`, `b` oraz rozwiązanie URL `A.LUsolve(b)` może być baaardzo czasochłonne...

Przykład: generowanie macierzy symbolicznie

```
>>> h, x = sym.symbols('h x')
>>> nodes = [0, h, 2*h]
>>> elements = [[0, 1], [1, 2]]
>>> phi = basis(d=1)
>>> f = x*(1-x)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=True)
>>> A
[h/3,   h/6,   0]
[h/6, 2*h/3, h/6]
[ 0,   h/6, h/3]
>>> b
[     h**2/6 - h**3/12]
[      h**2 - 7*h**3/6]
[5*h**2/6 - 17*h**3/12]
>>> c = A.LUsolve(b)
>>> c
[                                 h**2/6]
[12*(7*h**2/12 - 35*h**3/72)/(7*h)]
[  7*(4*h**2/7 - 23*h**3/21)/(2*h)]
```

Przykład: generowanie macierzy numerycznie

```
>>> nodes = [0, 0.5, 1]
>>> elements = [[0, 1], [1, 2]]
>>> phi = basis(d=1)
>>> x = sym.Symbol('x')
>>> f = x*(1-x)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=False)
>>> A
[ 0.166666666666667, 0.0833333333333333,                  0]
[0.0833333333333333,  0.333333333333333, 0.0833333333333333]
[                 0, 0.0833333333333333,  0.166666666666667]
>>> b
[          0.03125]
[0.104166666666667]
[          0.03125]
>>> c = A.LUsolve(b)
>>> c
[0.0416666666666666]
[ 0.291666666666667]
[0.0416666666666666]
```

Struktura macierzy współczynników

```
>>> d=1; N_e=8; Omega=[0,1]  # 8 linear elements on [0,1]
>>> phi = basis(d)
>>> f = x*(1-x)
>>> nodes, elements = mesh_symbolic(N_e, d, Omega)
>>> A, b = assemble(nodes, elements, phi, f, symbolic=True)
>>> A
[h/3,   h/6,     0,     0,     0,     0,     0,     0,   0]
[h/6, 2*h/3,   h/6,     0,     0,     0,     0,     0,   0]
[  0,   h/6, 2*h/3,   h/6,     0,     0,     0,     0,   0]
[  0,     0,   h/6, 2*h/3,   h/6,     0,     0,     0,   0]
[  0,     0,     0,   h/6, 2*h/3,   h/6,     0,     0,   0]
[  0,     0,     0,     0,   h/6, 2*h/3,   h/6,     0,   0]
[  0,     0,     0,     0,     0,   h/6, 2*h/3,   h/6,   0]
[  0,     0,     0,     0,     0,     0,   h/6, 2*h/3, h/6]
[  0,     0,     0,     0,     0,     0,     0,   h/6, h/3]
```

Uwaga (zadanie domowe): Wykonaj obliczenia na kartce papieru w celu potwierdzenia wartości poszczególnych elementów powyższej macierzy (pomocne w zrozumieniu materiału).

Wynik w przypadku ogólnym (N jednakowych elementów)

- Macierz rzadka -> większość współczynników to zera
- Przykład dla elementów typu P1, siatka regularna

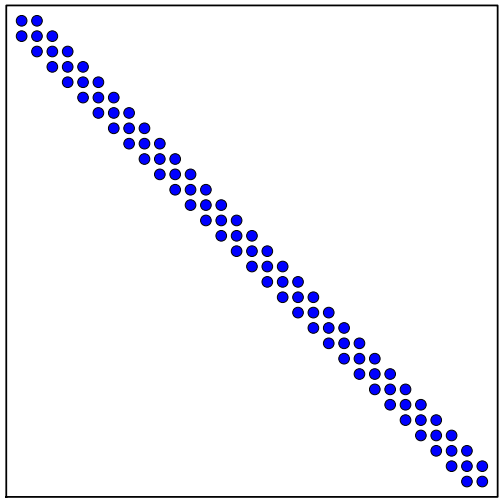
$$A = \frac{h}{6}\begin{pmatrix} 2 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 \\ 1 & 4 & 1 & \ddots & & & & & \vdots \\ 0 & 1 & 4 & 1 & \ddots & & & & \vdots \\ \vdots & \ddots & & \ddots & \ddots & 0 & & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & & 0 & 1 & 4 & 1 & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & & & & \ddots & 1 & 4 & 1 \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & 2 \end{pmatrix}$$

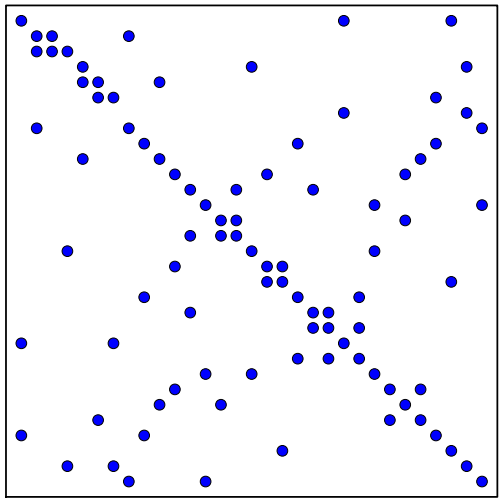
Macierz rzadka dla elementów typu P2 (siatka regularna)

$$A = \frac{h}{30} \begin{pmatrix} 4 & 2 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 16 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 2 & 8 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 16 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 2 & 8 & 2 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 16 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 2 & 8 & 2 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 & 16 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 2 & 4 \end{pmatrix}$$

Macierz rzadka dla siatek regularnych/indeksowanych losowo dla elementów P1

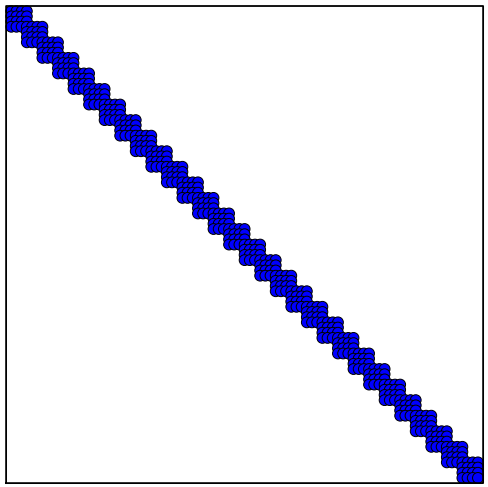
- Po lewej: węzły i elementy ideksowane od lewej do prawej
- Po prawej: węzły i elementy indeksowane "losowo"

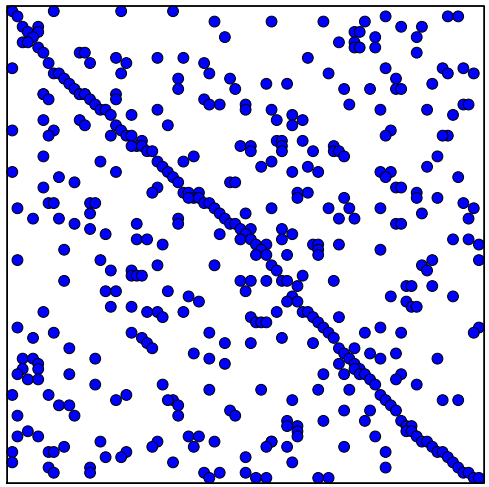




Macierz rzadka dla siatek regularnych/indeksowanych losowo dla elementów P3

- Po lewej: węzły i elementy ideksowane od lewej do prawej
- Po prawej: węzły i elementy indeksowane "losowo"





Macierze rzadkie – podsumowanie

Postać specyficznych macierzy $A_{i,j}$:

- Elementy P1: 3 niezerowe elementy w wierszu
- Elementy P2: 5 niezerowe elementy w wierszu
- Elementy P3: 7 niezerowe elementy w wierszu

Wskazówki:

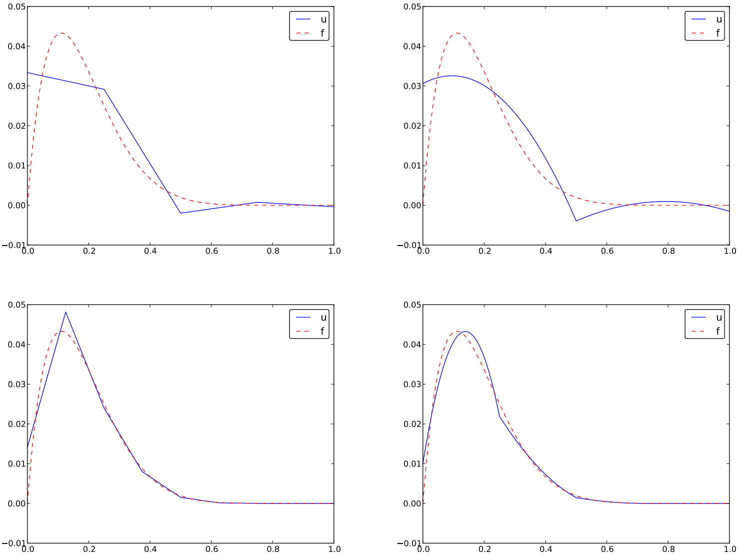
- Należy używać specjalne techniki przechowywania takich macierzy w pamięci i specjalnych solverów dla macierzy rzadkich
- W Pythonie: pakiet `scipy.sparse`

Przykład: przybliżenie funkcji $f \sim x^9$ elementami różnego typu; kod

Zadanie: Porównać rozwiązanie zadania przybliżenia funkcji $f(x)$ przy pomocy siatki N_e elementów skończonych o funkcjach bazowych rzędu d.

```
import sympy as sym
from fe_approx1D import approximate
x = sym.Symbol('x')

approximate(f=x*(1-x)**8, symbolic=False, d=1, N_e=4)
approximate(f=x*(1-x)**8, symbolic=False, d=2, N_e=2)
approximate(f=x*(1-x)**8, symbolic=False, d=1, N_e=8)
approximate(f=x*(1-x)**8, symbolic=False, d=2, N_e=4)
```

Przykład: przybliżenie funkcji $f \sim x^9$ elementami różnego typu; rysunki



Ograniczenia zaprezentowanego podejścia elementów skończonych

Najczęstsza interpretacja:

- *Węzły*: punkty potrzebne do zdefiniowania φ_i i obliczania

wartości u (wymagane do geometrii i aproksymacji funkcji)

- *Elementy*: podobszary (zawierające kilka węzłów)

Problem:

- węzły na brzegu potrzebne przy warunkach brzegowych, a nie zawsze tak musi być dla szczególnych rodzajów interpolacji (np. elementy stałe)
- Trzeba wymyśleć coś lepszego...

Uogólnienie koncepcji elementu skończonego (komórki, wierzchołki, węzły, stopnie swobody)

- Rozdzielenie aproksymacji geometrii (obszaru) od aproksymacji funkcji "nad" obszarem
- Nowe pojęcia: *komórka* (ang. cell) – podobszar, element, kawałek obszaru
- Komórka zbudowana jest z *wierzchołków* (ang. vertices <- vertex) – (krańców przedziału w 1D)
- *Węzły* (ang. nodes) p- punkty, w których należy wyznaczyć wartość poszukiwanej funkcji (nie muszą pokrywać się z wierzchołkami!, ale mogą...)
- *Stopnie swobody* (ang. degrees of freedom) – wielkości reprezentowane przez c_j (niewiadome w URL) -> najczęściej: wartości funkcji w węźle $\sum_{j \in \mathcal{I}_s} c_j \varphi_j(x_i) = c_i$

wierzchołki -> komórki -> interpolacja geometrii

węzły, stopnie swobody -> interpolacja funkcji

Pojęcie elementu skończonego

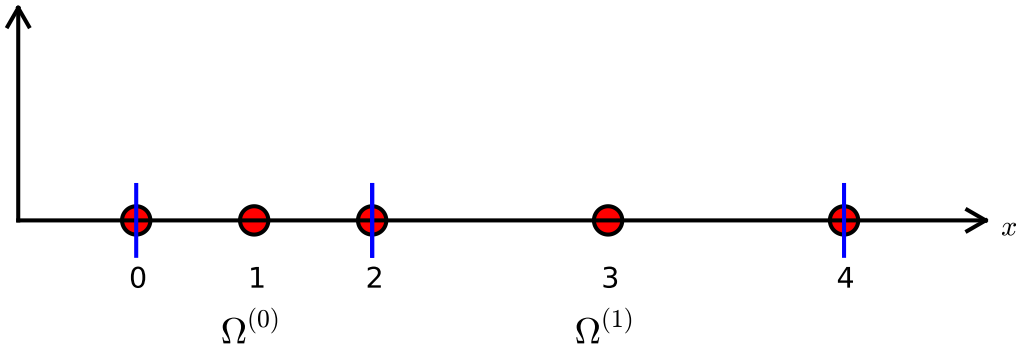
1. *komórka odniesienia* z unormowanym, lokalnym układem współrzędnych
2. zbiór *funkcji bazowych* $\tilde{\varphi}_r$ dla komórki
3. zbiór *stopni swobody* (t.j. wartości funkcji), jednoznacznie wyznaczający funkcje bazowe, dobrane tak aby $\tilde{\varphi}_r = 1$ dla r-tego stopnia swobody oraz $\tilde{\varphi}_r = 0$ dla wszystkich pozostałych stopni swobody
4. odwzorowanie (ang. mapping) pomiędzy lokalną a globalną indeksacją (transformacja numeracji) stopni swobody (*odwzorowanie dof – dof map*)
5. odwzorowanie komórki unormowanej na komórkę rzeczywistego obszaru (w 1D: $[-1, 1] \Rightarrow [x_L, x_R]$)

Struktury danych: vertices, cells, dof_map

- Współrzędne wierzchołków komórek: `vertices` (równoważne strukturze `nodes` dla elementów P1)
- Wierzchołki dla elementów (komórek): `cells[e][r]` numer globalny dla wierchołka `r` elementu `e` (równoważne strukturze `elements` dla elementów typu P1)
- `dof_map[e,r]` odwzorowanie lokalnego indeksu stopnia swobody `r` elementu `e` na number globalny (równoważne strukturze `elements` dla elementów typu Pd)

W trakcie assemblingu należy skorzystać ze struktury `dof_map`:

```
A[dof_map[e][r], dof_map[e][s]] += A_e[r,s]
b[dof_map[e][r]] += b_e[r]
```




```
vertices = [0, 0.4, 1]
cells = [[0, 1], [1, 2]]
dof_map = [[0, 1, 2], [2, 3, 4]]
```

Przykład: elementy P0

Przykład: Ta sama siatka, ale u to funkcja stała na każdej komórce (przedziałami stała) -> elementy typu P0.

Te same struktury `vertices` i `cells`, ale dodatkowo

```
dof_map = [[0], [1]]
```

Można traktować te elementy jak elementy z interpolacją opartą na węźle znajdującym się pośrodku elementu.

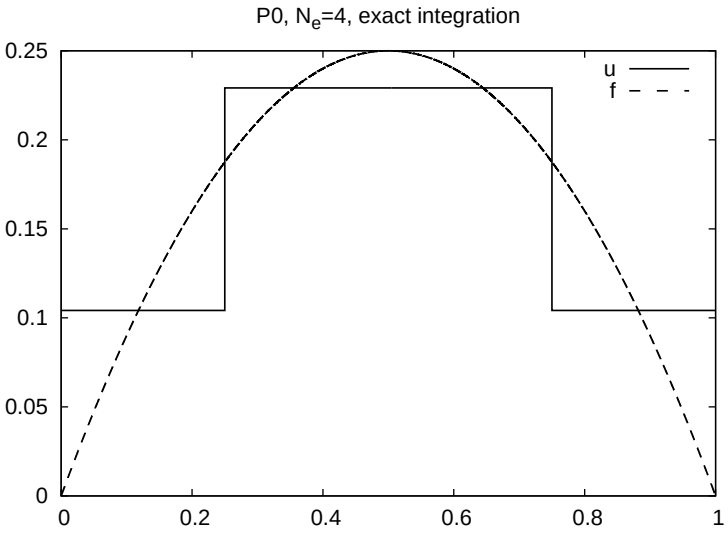
Uwaga:

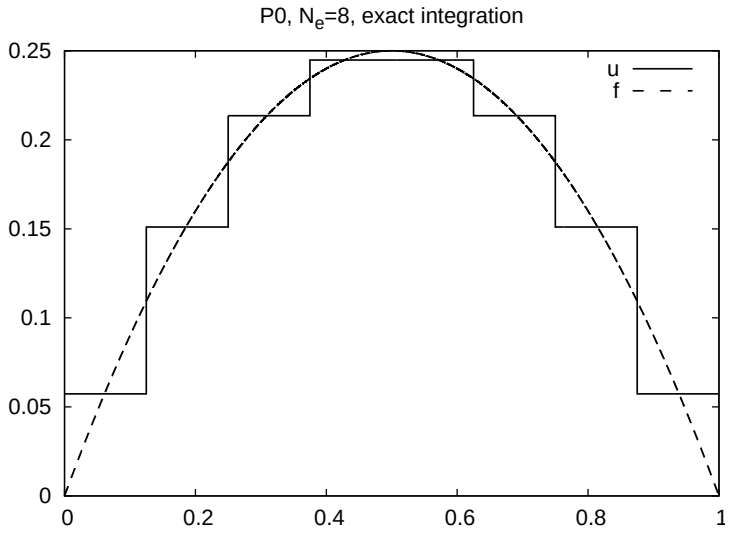
Od tej pory będziemy wykorzystywać struktury `cells`, `vertices`, i `dof_map`.

Szkielet programu

```
# Use modified fe_approx1D module
from fe_approx1D_numint import *

x = sym.Symbol('x')
f = x*(1 - x)

N_e = 10
# Create mesh with P3 (cubic) elements
vertices, cells, dof_map = mesh_uniform(N_e, d=3, Omega=[0,1])

# Create basis functions on the mesh
phi = [basis(len(dof_map[e])-1) for e in range(N_e)]

# Create linear system and solve it
A, b = assemble(vertices, cells, dof_map, phi, f)
c = np.linalg.solve(A, b)

# Make very fine mesh and sample u(x) on this mesh for plotting
x_u, u = u_glob(c, vertices, cells, dof_map,
                resolution_per_element=51)
plot(x_u, u)
```

Przybliżenie paraboli elementami P0





Funkcja approximate "opakowuje" polecenia z poprzedniego slajdu:

```
from fe_approx1D_numint import *
x=sym.Symbol("x")
for N_e in 4, 8:
    approximate(x*(1-x), d=0, N_e=N_e, Omega=[0,1])
```

Obliczanie błędów aproksymacji; uwagi ogólne

Błąd jako funkcja:

$$e(x) = f(x) - u(x)$$

Błąd – dyskretna wartość -> normy:

$$L^2 \text{ error:} \qquad ||e||_{L^2} = \left(\int_\Omega e^2 dx \right)^{1/2}$$

Szacowanie całki:

- dokładne, analityczne (symboliczne) - nieuniwersalne -> kwadratury
- odpowiednio dokładne spróbkowanie $u(x)$ w wielu punktach każdego elementu (np. poprzez wywołanie `u_glob`, które zwróci `x` i `u`), a następnie
- scałkowanie metodą trapezów
- Uwaga! Ważne! Całka powinna być policzona dokładnie "po elementach" (zmienność funkcji f)

Obliczanie błędów aproksymacji; szczegóły

Uwaga

Ponieważ elementy mogą być różnych rozmiarów (długości) siatka dyskretna może być niejednorodna, (ponadto powtórzone punkty na granicach elementów, widziane z perspektywy dwóch sąsiadujących elementów)

->

potrzebna prymitywna implementacja wzoru trapezów:

$$\int_\Omega g(x)dx \approx \sum_{j=0}^{n-1} \frac{1}{2}(g(x_j) + g(x_{j+1}))(x_{j+1} - x_j)$$

```
# Given c, compute x and u values on a very fine mesh
x, u = u_glob(c, vertices, cells, dof_map,
              resolution_per_element=101)
# Compute the error on the very fine mesh
e = f(x) - u
e2 = e**2
# Vectorized Trapezoidal rule
E = np.sqrt(0.5*np.sum((e2[:-1] + e2[1:])*(x[1:] - x[:-1])))
```

Zależność błędu od h i d

Teoria i eksperymenty pokazują, że aplikacja MNK czy metody Galerkina dla elementów skończonych typu Pd o tej samej długości h daje błąd:

$$||e||_{L^2} = C|f^{d+1}|h^{d+1}$$

gdzie C zależy od d i $\Omega = [0, L]$ ale nie zależy od h, oraz

$$|f^{d+1}|^2 = \int_0^L \left(\frac{d^{d+1} f}{dx^{d+1}} \right)^2 dx$$

Kubiczne wielomiany Hermite'a - definicja

- Czy da się skonstruować $\varphi_i(x)$ z ciągłą pochodną? Tak!

Niech dana będzie unormowana komórka $[-1, 1]$ z dwoma węzłami $X = -1$ i $X = 1$. Stopnie swobody:

- 0: wartość funkcji w $X = -1$
- 1: wartość pierwszej pochodnej w $X = -1$
- 2: wartość funkcji w $X = 1$
- 3: wartość pierwszej pochodnej w $X = 1$

Uwzględnienie wartości pochodnych zadanej funkcji w węzłach jako stopni swobody zapewnia kontrolę ciągłości pochodnej.

Kubiczne wielomiany Hermite'a - wyprowadzenie

4 ograniczenia na $\tilde{\varphi}_r$ (1 dla stopnia swobody r, 0 dla pozostałych):

- $\tilde{\varphi}_0(X_{(0)}) = 1,\ \tilde{\varphi}_0(X_{(1)}) = 0,\ \tilde{\varphi}'_0(X_{(0)}) = 0,\ \tilde{\varphi}'_0(X_{(1)}) = 0$
- $\tilde{\varphi}'_1(X_{(0)}) = 1,\ \tilde{\varphi}'_1(X_{(1)}) = 0,\ \tilde{\varphi}_1(X_{(0)}) = 0,\ \tilde{\varphi}_1(X_{(1)}) = 0$
- $\tilde{\varphi}_2(X_{(1)}) = 1,\ \tilde{\varphi}_2(X_{(0)}) = 0,\ \tilde{\varphi}'_2(X_{(0)}) = 0,\ \tilde{\varphi}'_2(X_{(1)}) = 0$
- $\tilde{\varphi}'_3(X_{(1)}) = 1,\ \tilde{\varphi}'_3(X_{(0)}) = 0,\ \tilde{\varphi}_3(X_{(0)}) = 0,\ \tilde{\varphi}_3(X_{(1)}) = 0$

Cztery układy równań liniowych z 4 niewiadomymi - współczynnikami wielomianów 3 stopnia.

Kubiczne wielomiany Hermite'a - wynik

$$\tilde{\varphi}_0(X) = 1 - \frac{3}{4}(X+1)^2 + \frac{1}{4}(X+1)^3 \tag{19}$$

$$\tilde{\varphi}_1(X) = -(X+1)(1 - \frac{1}{2}(X+1))^2 \tag{20}$$

$$\tilde{\varphi}_2(X) = \frac{3}{4}(X+1)^2 - \frac{1}{2}(X+1)^3 \tag{21}$$

$$\tilde{\varphi}_3(X) = -\frac{1}{2}(X+1)(\frac{1}{2}(X+1)^2 - (X+1)) \tag{22}$$

$$\tag{23}$$

Kubiczne wielomiany Hermite'a - sprawdzenie

```
# definition of the interval ends
x = np.array([-1, 1])

C = []  # list of polynomials stored as coefficients
B = []  # list of basis functions
dB = [] # list of the derivatives of basis functions

for k in np.arange(0,4):
    A = np.array(  [[   x[0]**3,   x[0]**2, x[0], 1 ],
                   [ 3*x[0]**2, 2*x[0],       1, 0 ],
                   [   x[1]**3,   x[1]**2, x[1], 1 ],
                   [ 3*x[1]**2, 2*x[1],       1, 0 ]])

    b = np.zeros( (4,1) ); b[k] = 1

    c = np.linalg.solve(A, b); C.append( c )

    B.append( lambda x:     C[k][0,0] * x**3 +   C[k][1,0] * x**2 + C[
    dB.append( lambda x: 3* C[k][0,0] * x**2 + 2*C[k][1,0] * x + C[k][
```

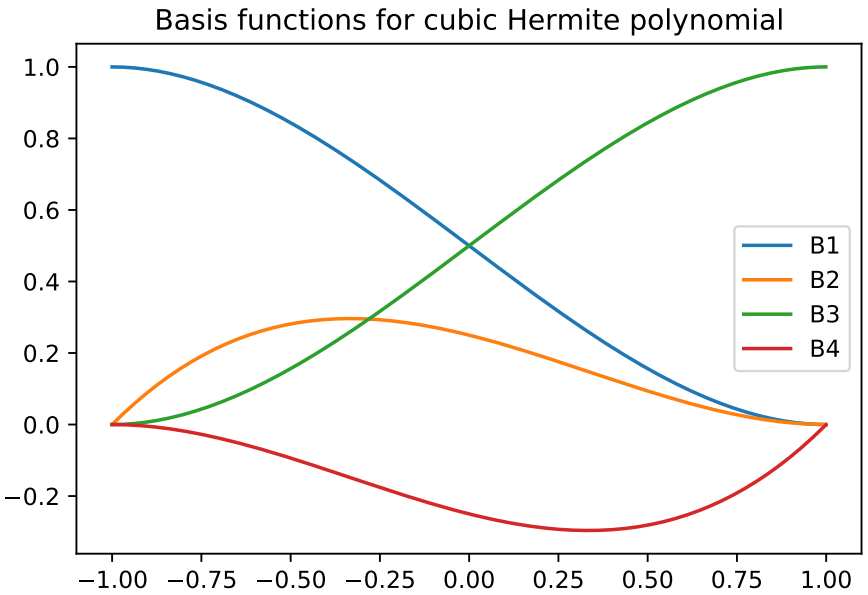
Kubiczne wielomiany Hermite'a - sprawdzenie

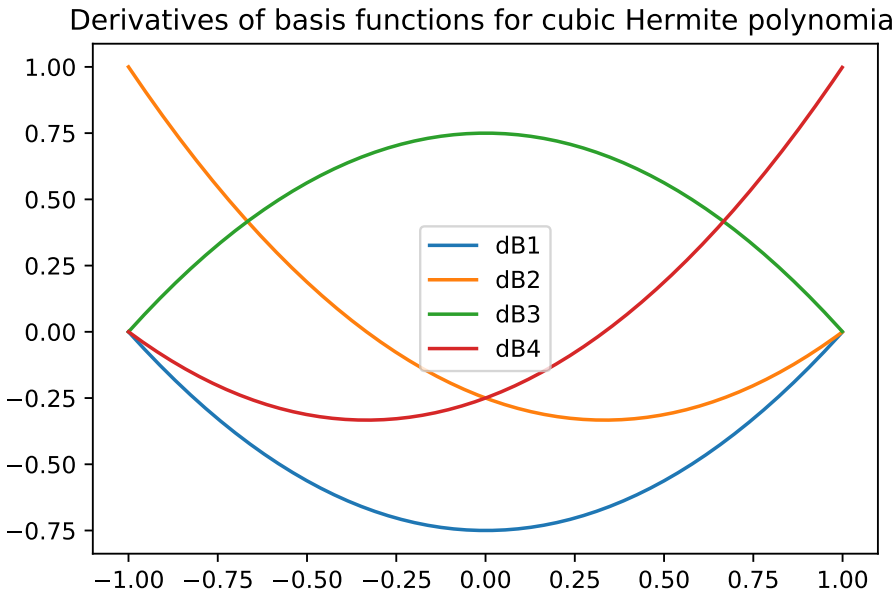
```
# Check numerically that resulting cubic polynomial
# fulfills imposed requirements
A = [1,       1,       2,      1]       # basis function coefficients
#    U(x[0])  dU(x[0]) U(x[1]) dU(x[1])

xx = np.arange(-1,1, 0.001)
U = np.zeros(xx.shape)

for k in np.arange(0,4):
    U = U + A[k] * B[k](xx)

# numerical approximation of the derivatives at the ends of the interv
dl = (U[1]-U[0])/(xx[1]-xx[0])
dr = (U[-1]-U[-2])/(xx[-1]-xx[-2])

numericalApproximationOfA = [ U[0], dl, U[-1], dr]

print(numericalApproximationOfA)
```

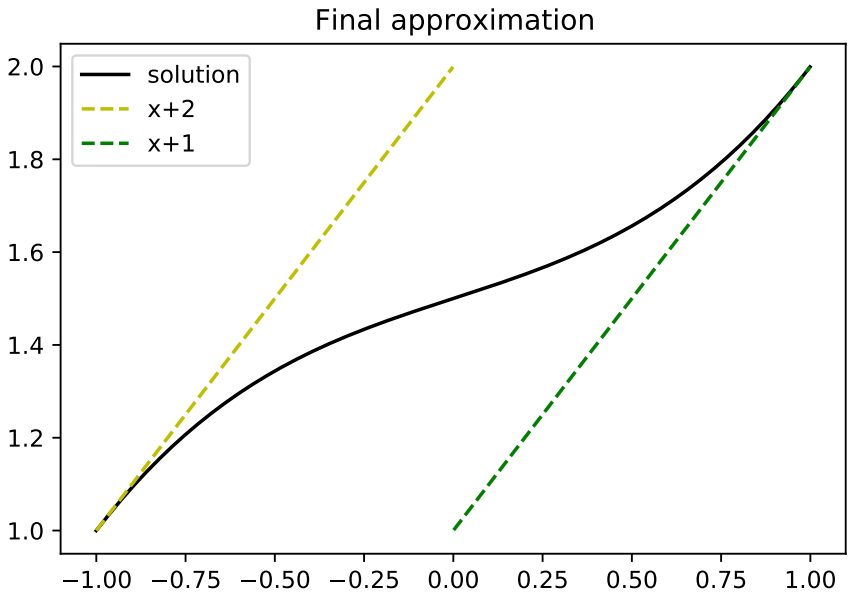
Wynik działania skryptu:

```
[1.0, 0.999250250000269, 1.999000749750002, 0.997751750000526]
```

Kubiczne wielomiany Hermite'a - wyniki







Całkowanie numeryczne

- $\int_\Omega f \varphi_i dx$ - konieczność całkowania numerycznego
- Współczynniki macierzy lewej strony - zwykle również numerycznie (bo wygodnie)

Ogólna postać kwadratury

$$\int_{-1}^{1} g(X)dX \approx \sum_{j=0}^{M} w_j g(\bar{X}_j),$$

gdzie

- $\bar{X}_j$ to *węzły kwadratury*
- w_j – *wagi kwadratury*

Różne metody -> różny wybór węzłów i wag

Wzór prostokątów

(ang. midpoint rule) – metoda punktu środkowego

Najprostsza metoda

$$\int_{-1}^{1} g(X)dX \approx 2g(0), \quad \bar{X}_0 = 0, \; w_0 = 2,$$

Dokładna dla funkcji podcałkowych będących wielomianami 1. stopnia

Metody Newtona-Cotesa

- Idea: węzły kwadratury równomiernie rozmieszczone na $[-1, 1]$
- Węzły kwadratury często pokrywają się węzłami siatki

Wzór trapezów:

$$\int_{-1}^{1} g(X)dX \approx g(-1) + g(1), \quad \bar{X}_0 = -1,\ \bar{X}_1 = 1,\ w_0 = w_1 = 1,$$

Wzór Simpsona (parabol):

$$\int_{-1}^{1} g(X)dX \approx \frac{1}{3}\left(g(-1) + 4g(0) + g(1)\right),$$

gdzie

$$\bar{X}_0 = -1,\ \bar{X}_1 = 0,\ \bar{X}_2 = 1,\ w_0 = w_2 = \frac{1}{3},\ w_1 = \frac{4}{3}$$

Metoda Gaussa-Legendre'a

- optymalne położenie węzłów kwadratury -> wyższa dokładność
- Kwadratury Gaussa-Legendre'a -> dobranie położenia węzłów oraz wag tak, aby całkować z jak najlepszą dokładnością

$$M = 1: \quad \bar{X}_0 = -\frac{1}{\sqrt{3}}, \ \bar{X}_1 = \frac{1}{\sqrt{3}}, \ w_0 = w_1 = 1 \tag{24}$$

$$M = 2: \quad \bar{X}_0 = -\sqrt{\frac{3}{5}}, \ \bar{X}_0 = 0, \ \bar{X}_2 = \sqrt{\frac{3}{5}}, \ w_0 = w_2 = \frac{5}{9}, \ w_1 = \frac{8}{9} \tag{25}$$

- $M = 1$: dokładna dla wielomianów 3. stopnia
- $M = 2$: dokładna dla wielomianów 5. stopnia
- W ogólności, M-punktowy wzór Gaussa-Legendre'a jest dokładny dla wielomianów stopnia $2M + 1$.

Plik 'numint.py zawiera zbiór węzłów i wag dla metody Gaussa-Legendre'a.

Aproksymacja funkcji w 2D

Rozwinięcie podejścia z 1D

Rozwiązania i algorytmy przedstawione dla aproksymacji funkcji $f(x)$ w 1D da się rozwinąć i "przenieść" na przypadki funkcji $f(x, y)$ w 2D i $f(x, y, z)$ w 3D. Ogólne wzory pozostają takie same.

Krótkie omówienie zagadnienia w 2D

Iloczyn skalarny w 2D:

$$(f, g) = \int_{\Omega} f(x, y) g(x, y) dx dy$$

Zastosowanie MNK lub metody Galerkina da URL:

$$\sum_{j \in \mathcal{I}_s} A_{i,j} c_j = b_i, \quad i \in \mathcal{I}_s$$

$$A_{i,j} = (\psi_i, \psi_j)$$

$$b_i = (f, \psi_i)$$

Problem: Jak skonstruować dwuwymiarowe funkcje bazowe $\psi_i(x, y)$?

Funkcje bazowe 2D jako iloczyn tensorowy funkcji 1D

Korzystając z funkcji bazowych 1D zmiennej x oraz funkcji bazowych 1D zmiennej y:

$$V_x = \text{span}\{\hat{\psi}_0(x), \ldots, \hat{\psi}_{N_x}(x)\} \tag{26}$$

$$V_y = \text{span}\{\hat{\psi}_0(y), \ldots, \hat{\psi}_{N_y}(y)\} \tag{27}$$

Przestrzeń wektorowa 2D może być zdefinowana jako *iloczyn tensorowy* $V = V_x \otimes V_y$ z funkcjami bazowymi:

$$\psi_{p,q}(x,y) = \hat{\psi}_p(x)\hat{\psi}_q(y) \quad p \in \mathcal{I}_x, q \in \mathcal{I}_y \,.$$

Iloczyn tensorowy

Niech dane będą dwa wektory $a = (a_0, \ldots, a_M)$ i $b = (b_0, \ldots, b_N)$. Ich *zewnętrznym iloczynem tensorowym* (*iloczynem diadycznym* jeśli $N = M$) jest $p = a \otimes b$ zdefiniowane jako:

$$p_{i,j} = a_i b_j, \quad i = 0, \ldots, M, \; j = 0, \ldots, N\,.$$

Uwaga: p to macierz/tablica dwuwymiarowa

Przykład: baza 2D jako iloczyn tensorowy przestrzeni 1D:

$$\psi_{p,q}(x, y) = \hat{\psi}_p(x)\hat{\psi}_q(y), \quad p \in \mathcal{I}_x, q \in \mathcal{I}_y$$

iloczyn tensorowy macierzy (dowolnych wymiarów) ->

iloczyn tensorowy wektorów (dowolnych wymiarów) ->

iloczyn diadyczny (tego samego wymiaru)

tensor

Równoważność notacji z dwoma lub jednym indeksem

Baza przestrzeni 2D wymaga dwóch indeksów (i podwójnego sumowania) :

$$u = \sum_{p \in \mathcal{I}_x} \sum_{q \in \mathcal{I}_y} c_{p,q} \psi_{p,q}(x, y)$$

Lub tylko jednego indeksu

$$u = \sum_{j \in \mathcal{I}_s} c_j \psi_j(x, y)$$

jeśli posiadamy odwzorowanie $(p, q) \to i$:

$$\psi_i(x, y) = \hat{\psi}_p(x) \hat{\psi}_q(y), \quad i = p(N_y + 1) + q \text{ or } i = q(N_x + 1) + p$$

Przykładowa baza przestrzeni 2D; wzory

Dla dwuelementowej bazy 1D

$$\{1, x\}$$

iloczyn tensorowy (wszystkie kombinacje) generuje bazę przestrzeni 2D:

$$\psi_{0,0} = 1, \quad \psi_{1,0} = x, \quad \psi_{0,1} = y, \quad \psi_{1,1} = xy$$

W notacji jednoindeksowej:

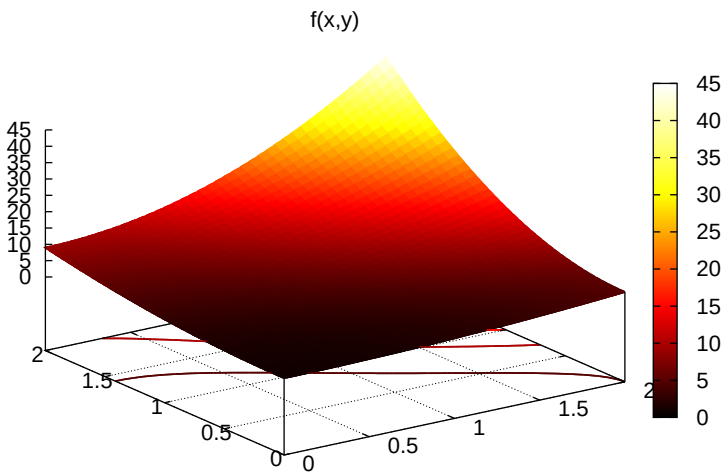
$$\psi_0 = 1, \quad \psi_1 = x, \quad \psi_2 = y, \quad \psi_3 = xy$$

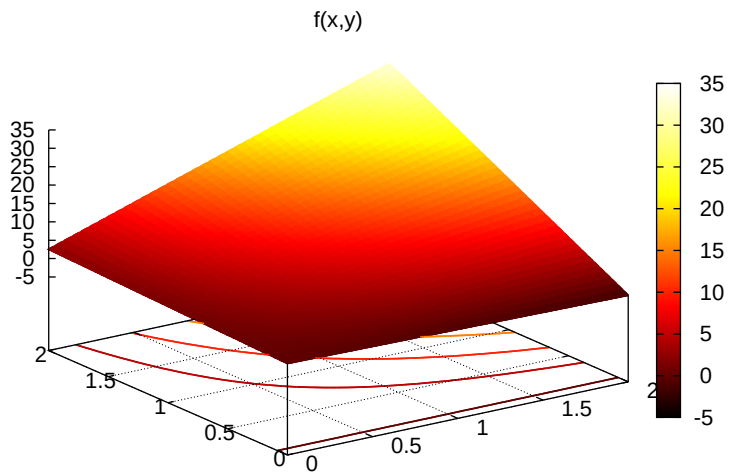
Przykładowa baza przestrzeni 2D; zastosowanie

$$\psi_0 = 1, \quad \psi_1 = x, \quad \psi_2 = y, \quad \psi_3 = xy$$

$$\int_0^{L_y} \int_0^{L_x} \left\{ \begin{bmatrix} 1\cdot 1 & 1\cdot x & 1\cdot y & 1\cdot xy \\ x\cdot 1 & x\cdot x & x\cdot y & x\cdot xy \\ y\cdot 1 & y\cdot x & y\cdot y & y\cdot xy \\ xy\cdot 1 & xy\cdot x & xy\cdot y & xy\cdot xy \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1\cdot f(x,y) \\ x\cdot f(x,y) \\ y\cdot f(x,y) \\ xy\cdot f(x,y) \end{bmatrix}$$

Funkcja aproksymowana (kwadratowa) $f(x,y) = (1+x^2)(1+2y^2)$ (po lewej), funkcja aproksymująca (biliniowa) u (po prawej) (x^2y^2 vs xy):





Implementacja; principal changes to the 1D code

Zmiany w kodzie w stosunku do wersji 1D (`approx1D.py`):

- `Omega = [[0, L_x], [0, L_y]]`
- Całkowanie symboliczne w 2D
- Generowanie funkcji bazowych 2D (jako iloczynów tensorowych)

Implementacja 2D: całkowanie

```
import sympy as sym

integrand = psi[i]*psi[j]
I = sym.integrate(integrand,
                  (x, Omega[0][0], Omega[0][1]),
                  (y, Omega[1][0], Omega[1][1]))

# Fall back on numerical integration if symbolic integration
# was unsuccessful
if isinstance(I, sym.Integral):
    integrand = sym.lambdify([x,y], integrand)
    I = sym.mpmath.quad(integrand,
                        [Omega[0][0], Omega[0][1]],
                        [Omega[1][0], Omega[1][1]])
```

Implementacja 2D: funkcje bazowe

Iloczyn tensorowy bazy potęgowej x^i (bazy Taylora):

```
def taylor(x, y, Nx, Ny):
    return [x**i*y**j for i in range(Nx+1) for j in range(Ny+1)]
```

Iloczyn tensorowy bazy sinusoidalnej $\sin((i+1)\pi x)$:

```
def sines(x, y, Nx, Ny):
    return [sym.sin(sym.pi*(i+1)*x)*sym.sin(sym.pi*(j+1)*y)
            for i in range(Nx+1) for j in range(Ny+1)]
```

Cały kod w `approx2D.py`.

Implementacja 2D: zastosowanie

$$f(x,y) = (1+x^2)(1+2y^2)$$

```
>>> from approx2D import *
>>> f = (1+x**2)*(1+2*y**2)
>>> psi = taylor(x, y, 1, 1)
>>> Omega = [[0, 2], [0, 2]]
>>> u, c = least_squares(f, psi, Omega)
>>> print u
8*x*y - 2*x/3 + 4*y/3 - 1/9
>>> print sym.expand(f)
2*x**2*y**2 + x**2 + 2*y**2 + 1
```

Implementacja 2D: przykład zastosowanie bazy umożliwiającej konstrukcję rozwiązania dokładnego

Dodajemy funkcje bazowe o wyższych potęgach tak, aby $f \in V$.
Spodziewany wynik: $u = f$

```
>>> psi = taylor(x, y, 2, 2)
>>> u, c = least_squares(f, psi, Omega)
>>> print u
2*x**2*y**2 + x**2 + 2*y**2 + 1
>>> print u-f
0
```

Uogólnienie do zagadnień 3D

Kluczowa idea:

$$V = V_x \otimes V_y \otimes V_z$$

Zastosowanie iloczynu tensorowego do wygenerowania bazy przestrzeni m wymiarowerj

$$a^{(q)} = (a_0^{(q)}, \ldots, a_{N_q}^{(q)}), \quad q = 0, \ldots, m$$

$$p = a^{(0)} \otimes \cdots \otimes a^{(m)}$$

$$p_{i_0, i_1, \ldots, i_m} = a_{i_1}^{(0)} a_{i_1}^{(1)} \cdots a_{i_m}^{(m)}$$

W szczególności dla 3D:

$$\psi_{p,q,r}(x, y, z) = \hat{\psi}_p(x)\hat{\psi}_q(y)\hat{\psi}_r(z)$$

$$u(x, y, z) = \sum_{p \in \mathcal{I}_x} \sum_{q \in \mathcal{I}_y} \sum_{r \in \mathcal{I}_z} c_{p,q,r} \psi_{p,q,r}(x, y, z)$$

Elementy skończone w 2D i 3D

Zalety FEM w zastosowaniach 2D i 3D:

- łatwość aproksymowania skomplikowanych geometrii
- łatwość generowania wielomianów (funkcji bazowych) wyższych rzędów w celu zwiększenia dokładności aproksymacji funkcji

FEM w 1D: głównie dla celów dydaktycznych, debugowania

Przykłady komórek 2D i 3D

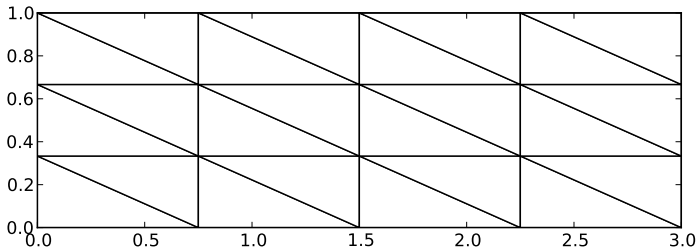
2D:

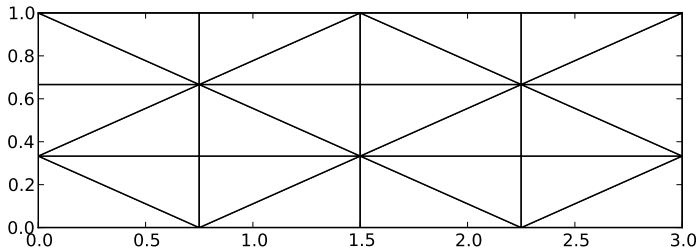
- trójkąty (triangles)
- czworokąty (quadrilaterals)

3D:

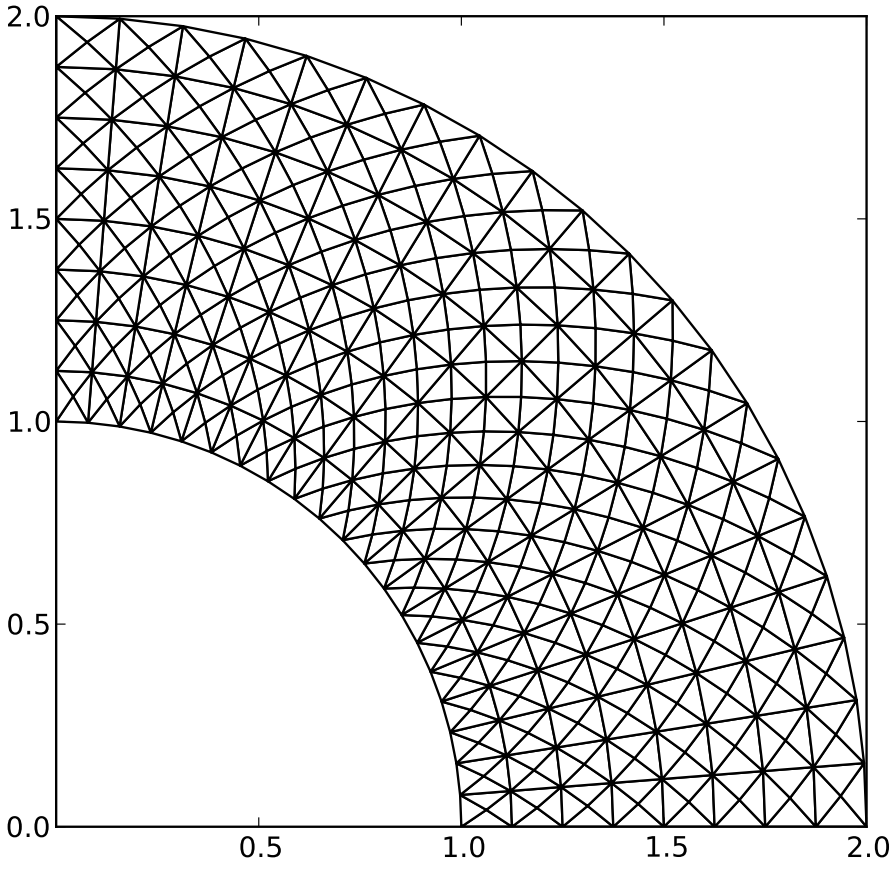
- czworościany (tetrahedra)
- sześciościany (hexahedra)

Obszar prostokątny (2D) zbudowany z elementów typu P1

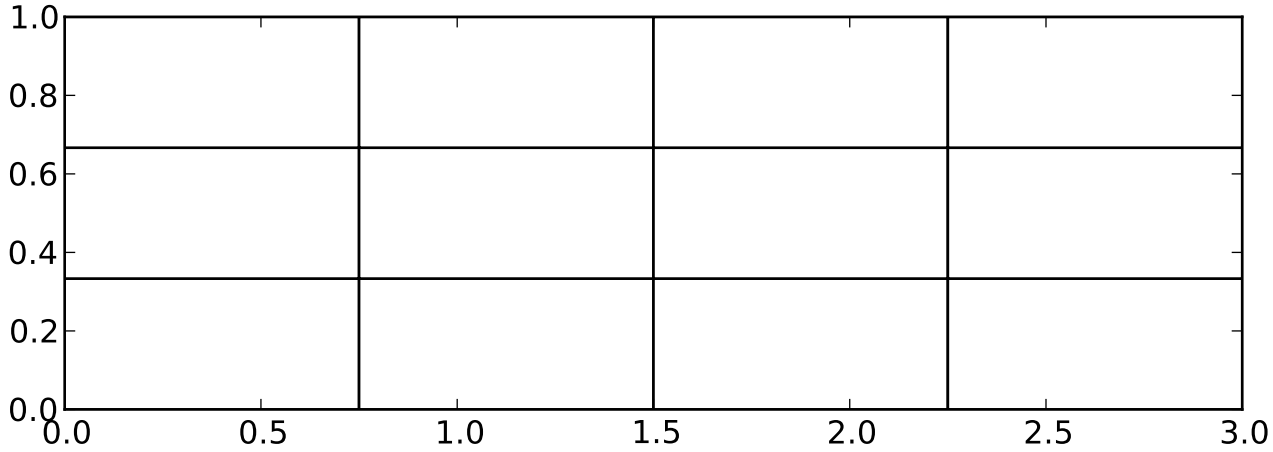




Nieregularny obszar 2D zbudowany z elementów typu P1

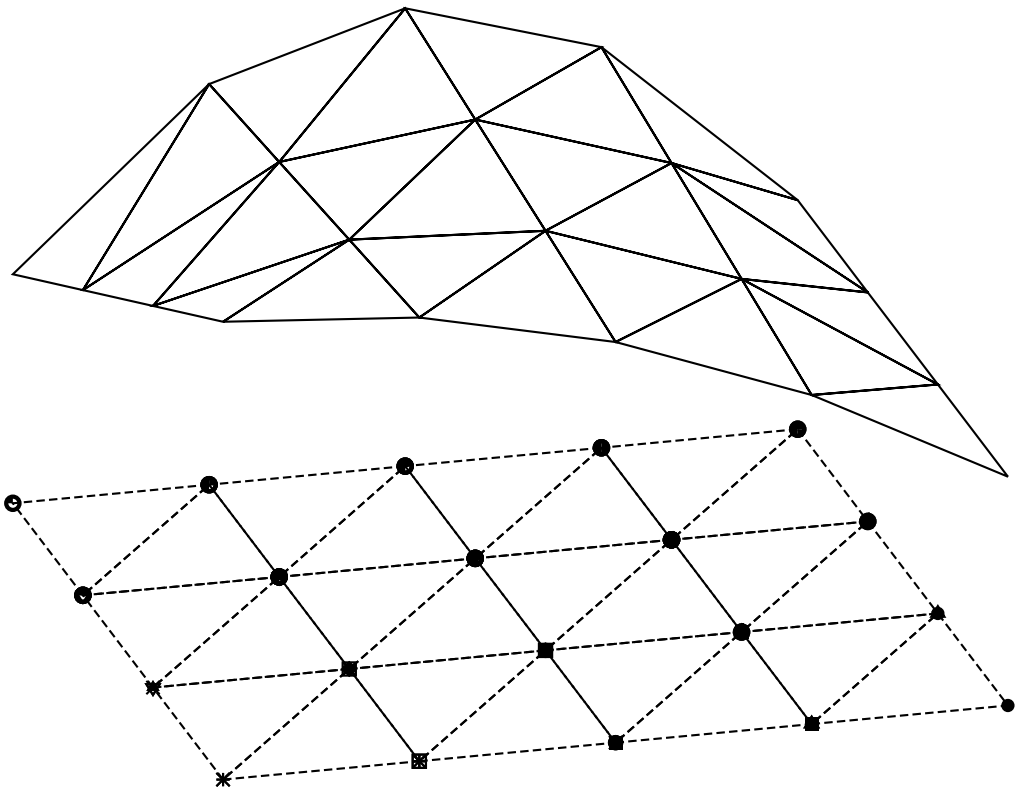


Obszar prostokątny (2D) zbudowany z elementów typu Q1



Aproksymacja funkcji 2D na siatce elementów trójkątnych

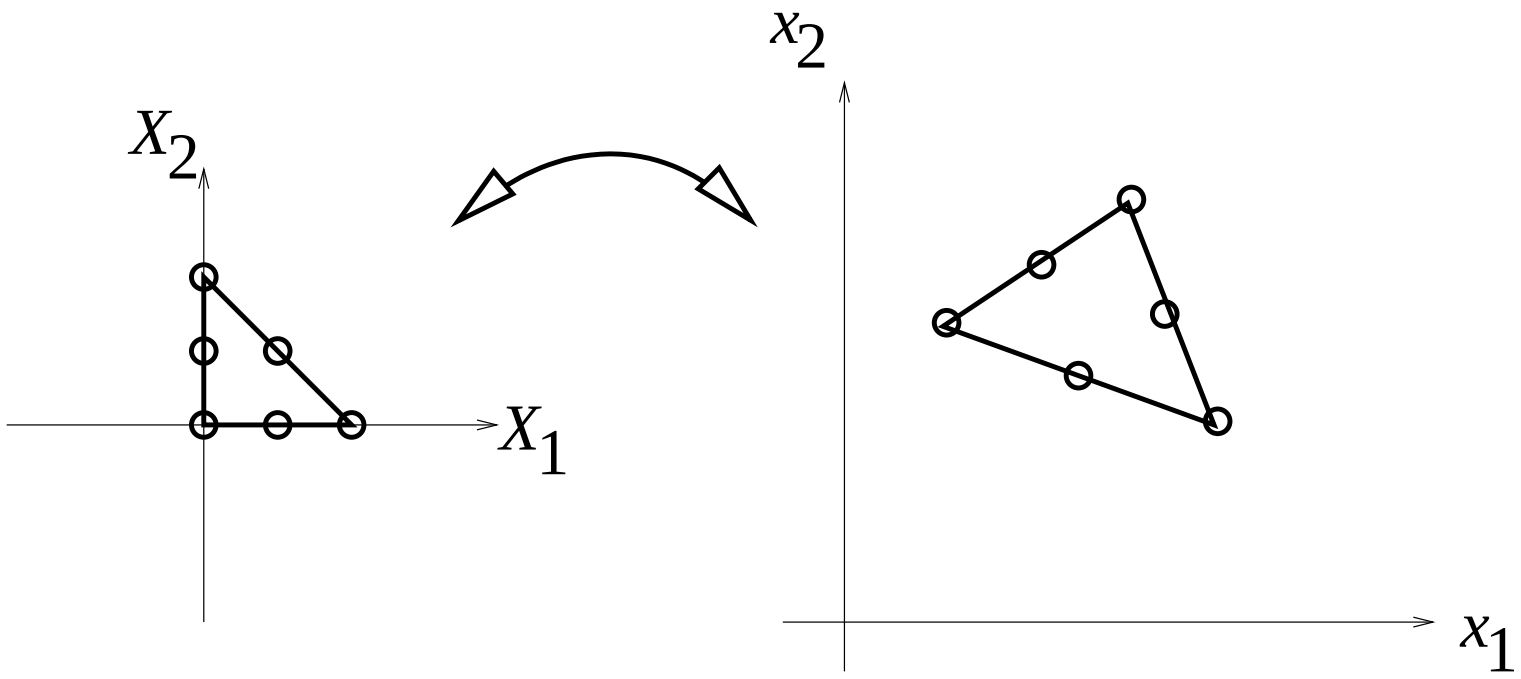
Element trójkątny typu P1: aproksymacja u na każdym elemencie (komórce) funkcją liniową $ax + by + c$



Własności elementów 2D typu P1

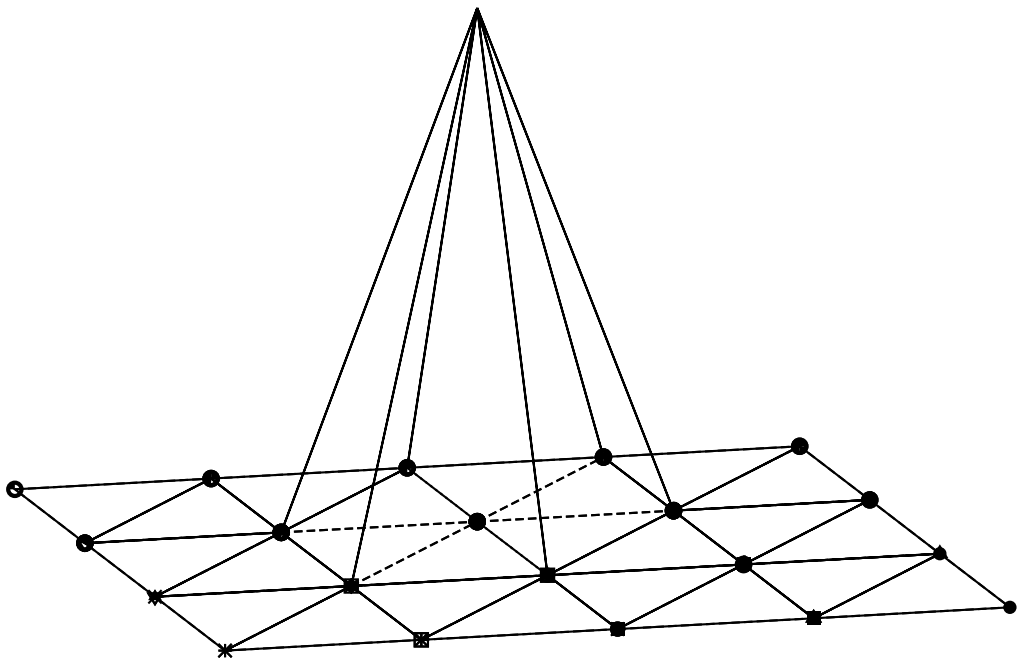
- Komórki = trójkąty
- Wierzchołki = wierzchołki komórek
- węzły = wierzchołki trójkąta
- Stopnie swobody = wartości funkcji w węzłach
- $\tilde{\varphi}_r(X, Y)$ jest funkcją liniową na komórce unormowanej
- $\varphi_i(x, y)$ jest odzorowaniem $\tilde{\varphi}_r(X, Y)$ na komórce rzeczywistej

Odwzorowanie liniowe elementu unormowanego na komórkę trójkątną



φ_i: funkcja-piramida

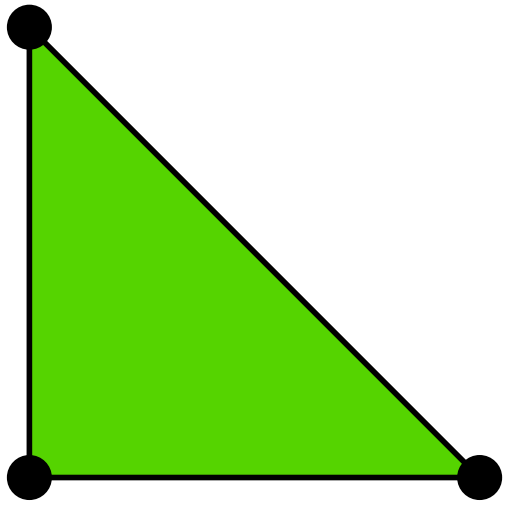
- $\varphi_i(x, y)$ – zmienność liniowa na poszczególnych komórkach
- $\varphi_i = 1$ w wierzchołku (węźle) i, 0 w pozostałych wierzchołkach (węzłach)



Elementy macierzy i wektora prawej strony

- Jak w 1D, wkład pojedynczej komórki do macierzy globalnego URL ogranicza się do kilku wartości w macierzy i wektorze wyrazów wolnych
- $\varphi_i \varphi_j \neq 0$ wtedy i tylko wtedy gdy i oraz j są stopniami swobody (wierzchołkami/węzłami) na tym samym elemencie
- Lokalna macierz trójkątengo elementu P1 to macierz o rozmiarach 3×3

Funkcje bazowe na unormowanym elemencie trójkątnym



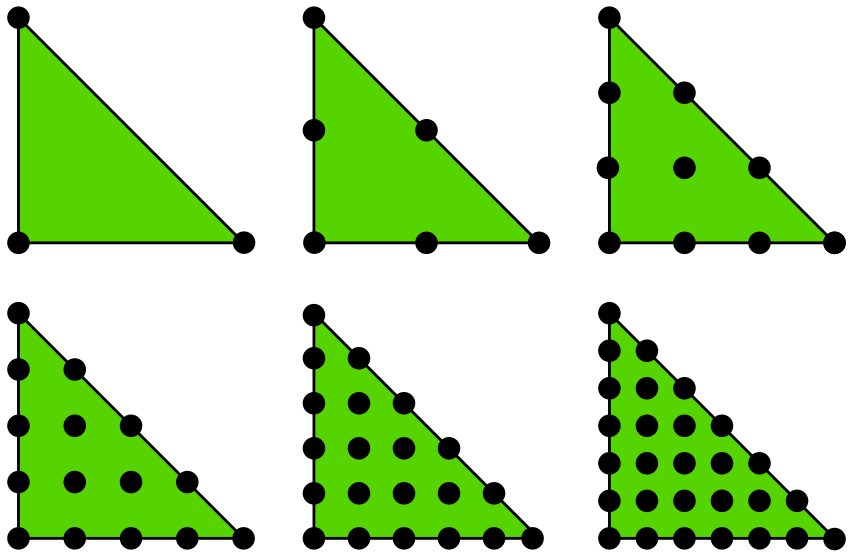
$$\tilde{\varphi}_0(X, Y) = 1 - X - Y \tag{28}$$

$$\tilde{\varphi}_1(X, Y) = X \tag{29}$$

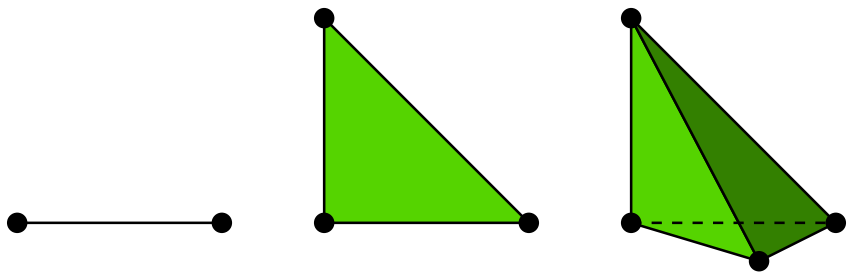
$$\tilde{\varphi}_2(X, Y) = Y \tag{30}$$

Funkcje bazowe $\tilde{\varphi}_r$ wyższych stopni opierają się na większej liczbie węzłów (stopni swobody)

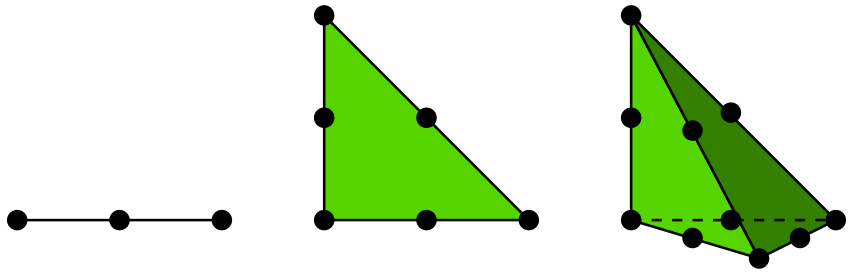
Elementy trójkątne typu P1, P2, P3, P4, P5, P6 przestrzeni 2D



Elementy P1 przestrzeni 1D, 2D i 3D



Elementy P2 przestrzeni 1D, 2D i 3D



- Interval, triangle, tetrahedron: *sympleksy* (ang. *simplex* -> *simplices*/*simplexes*)
- *ściana* (ang. *face*) – bok komórki (ścianka/krawedź/punkt)
- W czworościanie również *krawędzie* (*edges*)

Odwzorowanie afiniczne komórki unormowanej – wzór

Transformacja (Odwzorowanie) komórki we współrzędnych unormowanych

$$\boldsymbol{X} = (X, Y)$$

do komórki we współrzędnych globalnych:

$$\boldsymbol{x} = (x, y):$$

$$\boldsymbol{x} = \sum_r \tilde{\varphi}_r^{(1)}(\boldsymbol{X}) \boldsymbol{x}_{q(e,r)} \tag{31}$$

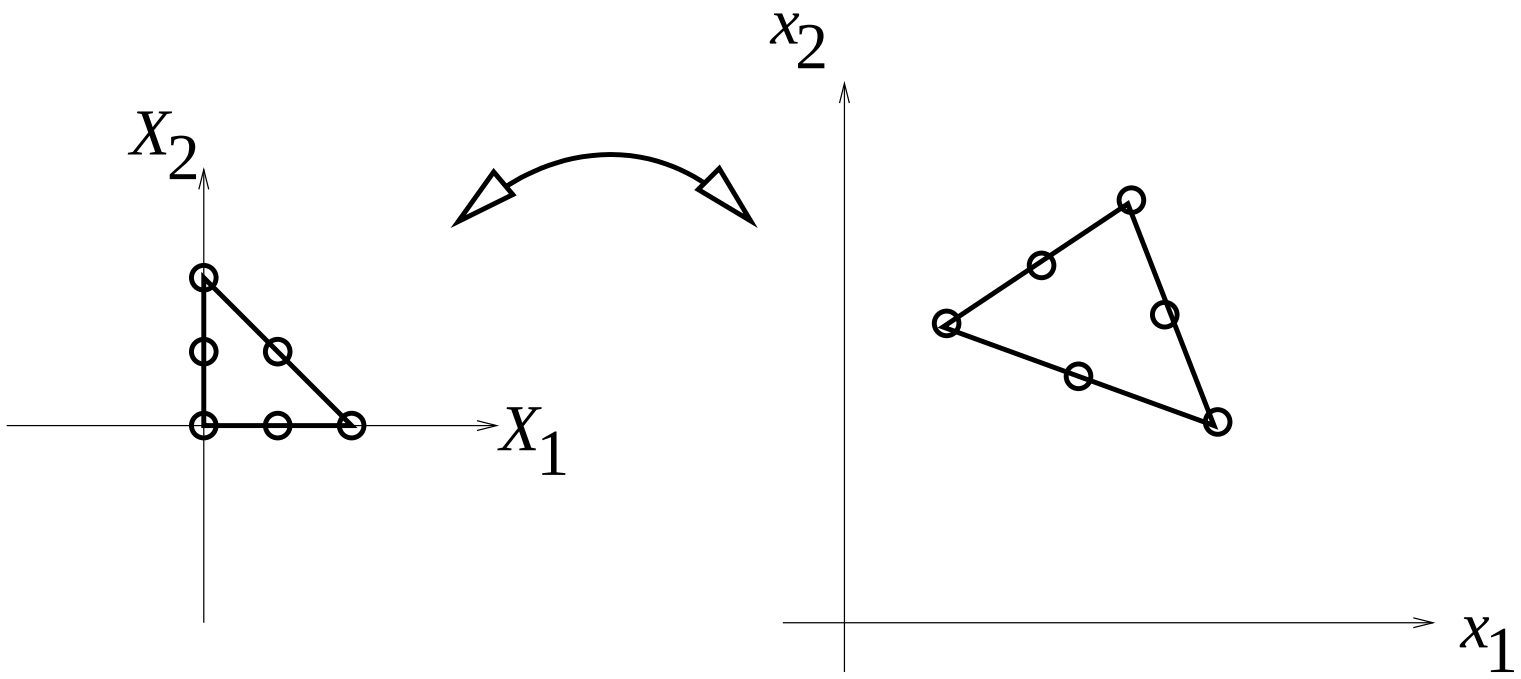
gdzie

- r przebiega przez wszystkie wierzchołki komórki
- $\boldsymbol{x}_i$ to globalne współrzędne (x, y) wierzchołka i
- $\tilde{\varphi}_r^{(1)}$ to funkcja bazowa typu P1

Odwzorowanie zachowuje liniowość ścian i krawędzi.

- TODO (Przykład rachunkowy)

Odwzorowanie afiniczne komórki unormowanej

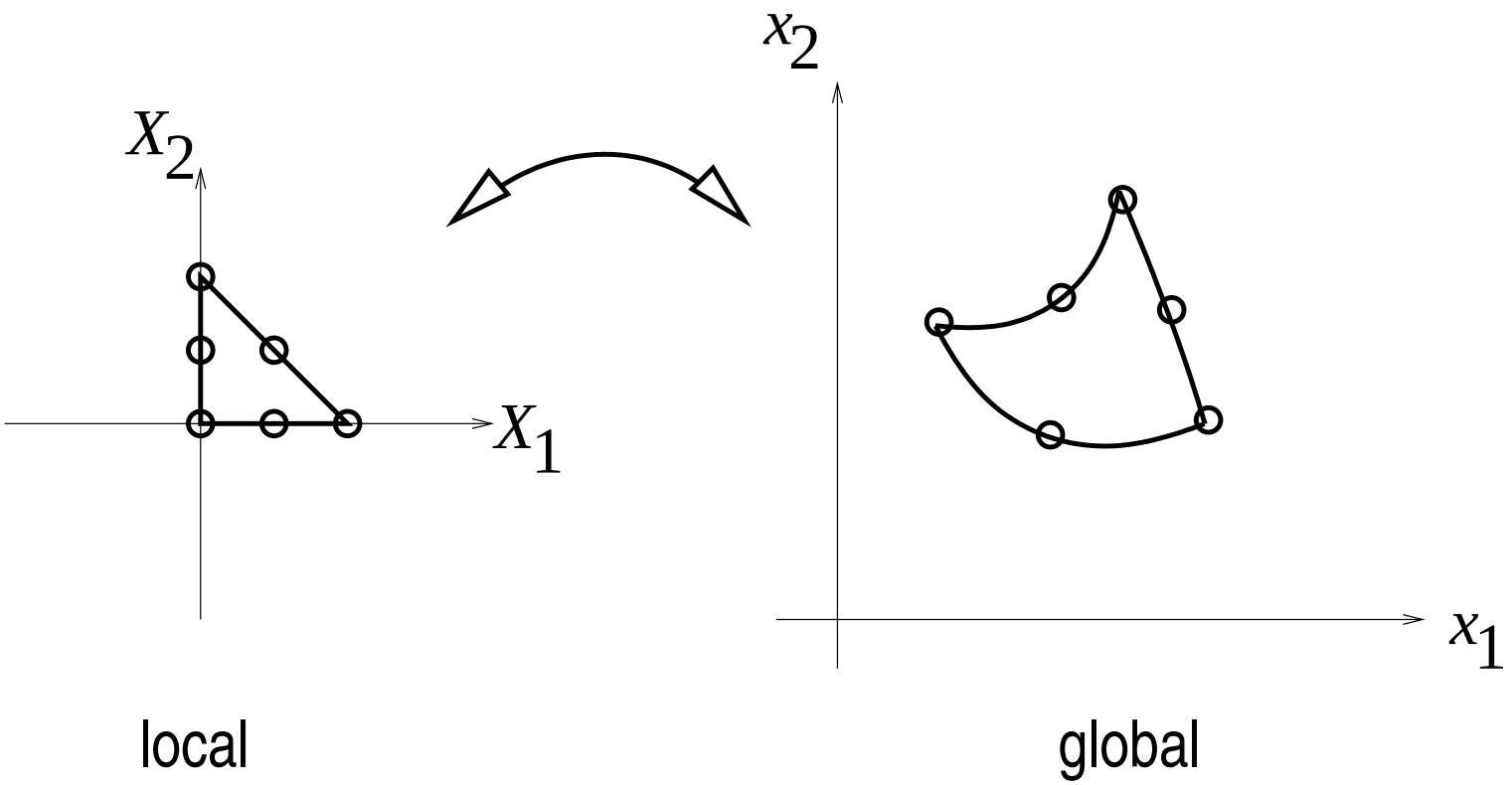


Komórki izoparametryczne

Idea: Wykorzystanie funkcji bazowych elementu (nie tylko funkcji typu P1 ale i wyższych rzędów) do odwzorowania geometrii:

$$\boldsymbol{x} = \sum_r \tilde{\varphi}_r(\boldsymbol{X}) \boldsymbol{x}_{q(e,r)}$$

Zaleta: pozwala generować elementy o geomtrii *nielinowej*



Obliczanie całek

Wymagana transformacja całek z $\Omega^{(e)}$ (komórka we współrządnych globalnych) w $\tilde{\Omega}^r$ (komórka unormowana/odniesienia):

$$\int_{\Omega^{(e)}} \varphi_i(\boldsymbol{x})\varphi_j(\boldsymbol{x})\,\mathrm{d}x = \int_{\tilde{\Omega}^r} \tilde{\varphi}_i(\boldsymbol{X})\tilde{\varphi}_j(\boldsymbol{X})\det J\,\mathrm{d}X \tag{32}$$

$$\int_{\Omega^{(e)}} \varphi_i(\boldsymbol{x})f(\boldsymbol{x})\,\mathrm{d}x = \int_{\tilde{\Omega}^r} \tilde{\varphi}_i(\boldsymbol{X})f(\boldsymbol{x}(\boldsymbol{X}))\det J\,\mathrm{d}X \tag{33}$$

gdzie $\mathrm{d}x = dxdy$ lub $\mathrm{d}x = dxdydz$ oraz $\det J$ to wyznacznik jakobianu odwzorowania $\boldsymbol{x}(\boldsymbol{X})$.

$$J = \begin{bmatrix} \frac{\partial x}{\partial X} & \frac{\partial x}{\partial Y} \\ \frac{\partial y}{\partial X} & \frac{\partial y}{\partial Y} \end{bmatrix}, \quad \det J = \frac{\partial x}{\partial X}\frac{\partial y}{\partial Y} - \frac{\partial x}{\partial Y}\frac{\partial y}{\partial X}$$

Odwzorowanie afiniczne (31): $\det J = 2\Delta$,
$\Delta =$ powierzchnia komórki/elementu

Uwaga dot. uogólnienia FEM z 1D do 2D/3D

Ogólna idea FEM oraz kroki algorytmu – takie same niezależnie od wymiarowości geometrii.
Im wyższy wymiar przestrzeni, tym większy nakład obliczeniowy. ze względu na komplikację wzorów.
Obliczenia ręczne - nużące, podatne na popełnienie pomyłki.
Automatyzacja i algorytmizacja problemu pożądana.