Mnożenie przez macież jako funkcja

Łatwo zauważyć, że w metodzie gradientów sprzężonych nie używamy elementów macierzy, lecz tylko możliwości mnożenia przez nią. Tzn: nie musimy wiedzieć jak wygląda A, wystarczy że dla danego wektora x potrafimy obliczyć Ax.

Na tych laboratoriach wykorzystamy tą wiedzę by dodatkowo przyspieszyć program i zmniejszyć użycie pamięci.

Przygotowanie

By nie pomylić się w następnych krokach, należy pierw dobrze ‘’posprzątać” kod.

Zadanie

Wydziel wszystkie elementy iteracji metody gradientów sprzężonych do oddzielnych pętli. Tak by $r = Ax$, $r = b - r$, etc. były oddzielnymi kawałkami kodu

Zadanie

Wydziel z funkcji `Solve` część odpowiedzialną za mnożenie przez A: `Mult(double** A, double*x, double* r)` i preconditioner diagonalny: `Precond(double** A, double*r, double* p)` — Zauważ że mnożenie przez macierz A występuje co najmniej dwa razy w iteracji.

Na tym etapie w funkcji `Solve` nie powinny występować nigdzie elementy macierzy A.

Zadanie

Przenieś zmienne `fix`, `thick` do zmiennych globalnych

Zadanie

Skopiuj funkcję `Mult` pod nazwą `SMult`

Element po elemencie

W funkcji `SMult` będziemy chcieli napisać funkcję mnożącą przez macierz sztywności nie używając samej macierzy S. Chcemy wykonać operację $r = Sx$, tzn: $r_i = \sum_j S_{ij} x_j$.

Jeśli dodamy do elementu $S_{1,2}$ liczbę 4, to do r_1 musimy dodać $4x_2$.

Analogicznie jeśli dodamy do elementu S_{ij} liczbę w, to tak jak byśmy dodali do elementu r_i liczbę $w \cdot x_j$. Jako, że macierz S konstruujemy właśnie przez dodawanie do kolejnych jej elementów, możemy całość mnożenia przez nią zapisać w powyższej postaci.

Zadanie

Przekopiuj fragment kodu funkcji `main` odpowiedzialny za konstrukcję macierzy S. Następnie, każde wystąpienie\ `S[i,j] += cos;`\ zamień na:\ `r[i] += cos * x[j];`

Co z częścią, która zamieniała wybrane wiersze na wiersze macierzy diagonalnej? Jeśli w macierzy S i-ty wiersz zamienimy na same zera i 1 na przekątnej, to tak jak byśmy postawili $r_i = x_i$.

Zadanie

Zamień pętlę wycinającą $i$ty wiersz, na `r[i]=x[i]`

Zadanie

Przetestuj kod z `SMult` zamiast `Mult`

Zadanie

Napisz trywialny preconditioner `IPrecond(double ** A, double * r, double * p)`, przepisujący $p = r$.

Zadanie

Popraw kod zauważając, że ani `SMult` ani `IPrecond` nie potrzebują brać `A` za argument.

A teraz na poważnie

Na tym etapie nigdzie w kodzie nie potrzebujemy macierzy S. Możemy ją całkowicie wyeliminować. Funkcję `Solve` będziemy chcieli jednak używać dla różnych macierzy — dlatego jako argument, zamiast macierzy `double ** A` będziemy przekazywać funkcję mnożenia `void (*mult)(double *, double *)`. Tzn: nagłówek funkcji `Solve` będzie następujący:\ `void Solve(int n, void (*mult)(double *, double *), double *b, double *x)`\ A w miejscu mnożenia przez macierz $r = Ax$ będziemy mieli `mult(x,r);`. Teraz funkcję `Solve` będziemy wywoływać przekazując jej konkretną funkcję mnożącą: `Solve(n, SMult, F, d);`.

Równoległość