Programowanie II R, ćwiczenia 8

Zadanie 1: (Dziedziczenie. Funkcja – Wielomian – Gauss)

Zaprogramuj klasę bazową `Funkcja` o publicznych polach `xmin` i `xmax`, reprezentujących krańce przedziału dziedziny funkcji. Wpisz konstruktor do ich ustawiania. Następnie dodaj czystą metodę wirtualną `double f (double)` : zapowie ona metody `f` w klasach pochodnych, które będą zwracać wartość danej funkcji.

Następnie zaprogramuj klasę `Wielomian`, dziedziczącą po klasie `Funkcja`. Niech będzie to wielomian o elastycznej liczbie stopni swobody, którego parametry przy potęgach zapisane będą np. w polu `vector<double> par`. Napisz konstruktor, przyjmujący krańce dziedziny oraz parametry wielomianu. Następnie napisz metodę `double f (double x)`, zwracającą wartość wielomianu dla zadanego `x`, o ile `x` mieści się w dziedzinie.

Zaimplementuj też klasę `Gauss`, dziedziczącą po klasie `Funkcja`, obsługującą funkcję `Gaussa` znormalizowaną do `1`, której polami będą double `x0` (centroid) oraz double `sigma` (dyspersja). Napisz konstruktor, przyjmujący krańce dziedziny oraz `x0` i `sigma`. Następnie napisz metodę `double f (double x)` , zwracającą wartość funkcji Gaussa dla zadanego `x`, o ile `x` mieści się w dziedzinie.

W funkcji `main()` :

– utwórz wielomian `W` o parametrach `{1, -4, 1}` w dziedzinie `[-10, 10]`. Wypisz na ekran `W(0)` i `W(1)` oraz zażądaj wypisania `W(-11)`.

– utwórz funkcję Gaussa `G` o parametrach {`x0 = 1`, `sigma = 1`} w dziedzinie `[-10, 10]`. Wypisz na ekran `G(0)` i `G(1)` oraz zażądaj wypisania `G(-11)`.

Na drugim etapie:

– dodaj klasie bazowej i pochodnym metodę `Status` (która wersja będzie wirtualna?). Jej celem będzie wypisanie wartości parametrów i przedziału funkcji. W funkcji `main()` zademonstruj działanie metody `Status()`.

– zmień metodę `double f (double)` na: `double operator() (double)`

Mówimy, że obiekt staje się „***funktorem***”, czyli obiektem wywoływalnym jak funkcja.

Zastosuj koncepcję metod wirtualnych tam, gdzie to potrzebne. Możesz oprogramować wyjątek, gdy żądany argument funkcji jest poza jej dziedziną.

– Poza klasami napisz funkcję `Wykres_terminal`, która w terminalu zaprezentuje w sposób uproszczony przebieg funkcji. Podziel dziedzinę na nieduże N przedziałów i wypróbkuj funkcję w środku każdego przedziału. Argument N oraz zakres rysowanych argumentów funkcji [xmin, xmax] niech funkcja przyjmuje w argumentach wejścia.

Wskazówka: Aby funkcja radziła sobie z obiektem z dowolnej klasy dziedziczącej, w argumentach wejścia podstaw `Funkcja& R` (referencję na obiekt klasy bazowej). Ponieważ będziesz potrzebował wartości funkcji dla danego `x`, to klasa bazowa `Funkcja` musi posiadać `operator()` w formie czystej metody wirtualnej:

```
virtual double operator() (double) = 0;
```

Zmodyfikuj też funkcję `main()` , aby zademonstrować powyższe usprawnienia.

Zadanie 2 (Wykres funkcji przy użyciu matplotlib-cpp)

Do kodu w zadaniu 1. dopisz funkcję

```
void Rysuj_do_pliku (Funkcja& Fun, double xmin, double xmax,
                                   double step, string fileName);
```

której zadaniem będzie narysowanie funkcji przejętej przez referencję `Fun`, jako zestawu punktów w zadanym przedziale i co dany krok. Rysunek powinien osiąść w pliku o zadanej nazwie.
Zwróć uwagę, że aby `Rysuj` mogło przyjąć dowolne obiekty funkcyjne Twoich klas, argumentem wejścia jest obiekt klasy bazowej: skorzystasz w ten sposób z *polimorfizmu klas*).
Dla pewności, kończąc rysowanie, zamknij obrazek poprzez `plt::close();` .

Do rysowania wykorzystaj nakładkę **matplotlib-cpp**. W funkcji `main()` wykonaj w ten sposób plik rysunkowy dla Twojego wielomianu w przedziale [-10, 10] oraz dla funkcji Gaussa w przedziale [-5, 5].

Wybór poleceń matplotlib-cpp, przydatnych do zadania tego i pokrewnych:

```
plt::plot    (vector<double> X, vector<double> Y) : rysuje punkty, domyślnie je łącząc
plt::scatter (vector<double> X, vector<double> Y,
                          double markersize = 1)  : rysuje punkty bez łączenia
plt::hist    (vector<double> X, long bins = 10)    : rysuje histogram danych w bins słupkach
plt::axvline (double X, double y0, double y1)      : pionowa przy X. Y to zakres osi [0,1]
plt::axhline (double Y, double y0, double y1)      : pionowa przy Y. X to zakres osi [0,1]
plt::xlim    (double xmin, double xmax)            : ustawia zakres osi X
plt::ylim    (double ymin, double ymax)            : ustawia zakres osi Y
plt::grid    (true)                                : w(y)łącza siatkę na wykresie
plt::show    ()                                    : pokazuje obrazek w ekranie graficznym
plt::save    ("obrazek.png")                       : zapisuje obrazek w formacie png (pdf)
plt::close   ()                                    : zamyka obrazek
```

W przypadku rysowania większej ilości danych, należy na wstępie kodu utworzyć nakładkę:
`plt::backend ("Agg");`

Styl ustawiamy, dodając do argumentów `plot` lub `scatter` – zbiór par {własność, wartość}, np. :
`plt::plot (X, Y, { {"color", "red"} , {"marker", "o"} , {"linestyle", "—"} } );`

Zadanie 3: (Dziedziczenie. Statystyka Boltzmanna / Bose-Einsteina / Fermiego-Diraca)

Napisz klasę bazową `Rozklad` o polach publicznych `double T` (reprezentuje temperaturę cząstki) i `string nazwa` oraz o konstruktorze pozwalającym zainicjować te pola.

Następnie zaprogramuj klasy dziedziczące publicznie klasę `Rozklad`, reprezentujące odpowiednie rozkłady: `Boltzmann` , `BoseEinstein` , `FermiDirac` . W każdej klasie zaprogramuj konstruktor do inicjowania pól. Ponadto oprogramuj `operator()` , który dla podanego argumentu `x` typu `double` zwraca wartość danego rozkładu fizycznego:

$$f_{Boltzmann}(x) = e^{-x/T} \qquad f_{Bose-Einstein}(x) = \frac{1}{e^{x/T} - 1} \qquad f_{Fermi-Dirac}(x) = \frac{1}{e^{x/T} + 1}$$

Dzięki temu operatorowi Twoje obiekty staną się *funktorami*.

(Uwaga: forma wzorów jest uproszczeniem oryginalnych rozkładów dla $\mu = 0$.
Możesz też zaprogramować formę pełniejszą, dodając nowe pole, np. `double mu` *)*

W funkcji `main` zadeklaruj 3 obiekty reprezentujące 3 powyższe rozkłady, nadając im w konstruktorze nazwę własną oraz wartość temperatury (np `T = 10` [jednostki umowne]) . Wypisz na ekran wartość każdego z obiektów dla `x = 5` .

Poza klasami napisz funkcję `Rysuj_mpl`, która naszkicuje dany rozkład przy użyciu interfejsu graficznego `matplotlibcpp.h` . Aby funkcja radziła sobie z obiektem z dowolnej klasy dziedziczącej, w argumentach wejścia pod przejęcie obiektu podstaw `Rozklad& R` (*polimorfizm klas*). Ponieważ będziesz potrzebować wartości rozkładu dla danych `x`, to klasa bazowa `Rozklad` musi posiadać `operator()` w formie *czystej metody wirtualnej*:

```
virtual double operator() (double) = 0;
```

Załóż, że szkicowanie ma objąć wartości `x` pomiędzy `xmin` a `xmax`, co zadany krok `dx`, które podawane są w argumentach wejścia funkcji `Rysuj_mpl`, np. `xmin = 0.2`, `xmax = 20`, `dx = 0.1`. Nazwę pliku graficznego również przekazuj przez argument wejścia funkcji.

W funkcji `main()` wywołaj rysowanie dla każdego z obiektów.