Pomoc do HL++ v.2.4

Spis treści

1 Spis funkcji HL++

1.1 Funkcje elementarne (*elfun*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Wartość bezwzględna	double dabs(double x)	dabs(-2.5)	Wartość bezwzględna liczby
Prawie zero	double nzero(double x)	nzero(0.0)	Zwraca EPS (1e-9), jeśli argument wynosi 0. W innych przypadkach zwraca 'x'. W przykładzie wynikiem jest 1e-9.
Praktycznie zero	int praczero(double x)	praczero(1e-30)	Zwraca 1 (true), jeśli argument jest nie większy od PREC (1e-15). W innych przypadkach zwraca 0 (false).
Praktycznie nie zero	int notzero(double x)	notzero(1e-30)	Zwraca negację praczero(x)
Praktycznie równe	int praceql(double x,double y)	praceql(1.0,1.01)	Porównuje 'x' i 'y' z dokładnością PREC (1e-15)
Praktycznie różne	int noteql(double x,double y)	noteql(1.0,1.01)	Zwraca negację praceql(x,y)
Znak liczby	double sign(double x)	sign(10.0)	Znak liczby
Maksimum	double max(double x,double	max(4,5)	Większa z liczb

	y) int max(int x,int y)		
Minimum	double min(double x,double y) int min(int x,int y)	min(4,5)	Mniejsza z liczb
Kwadrat liczby	double sqr(double x)	sqr(2.0)	Kwadrat liczby
Potęga całkowita	double powi(double x,int e)	powi(3.2,5)	Potęga całkowita liczby rzeczywistej
Liczba losowa	double rnd()	rnd()	Liczba losowa z rozkładu jednostajnego z przedziału domkniętego [0,1]
Liczba losowa 2	double rnd2()	rnd2()	Liczba losowa z rozkładu jednostajnego z przedziału otwartego (0,1)
Zaokrąglenie	double round(double x) double round(double x,int prec)	round(2.8)	Zaokrąglenie liczby. Można określić liczbę cyfr po przecinku ('prec'), które będą zaokrąglone
Czy całkowita	int is_int(double x)	is_int(8.5)	Zwraca 1 (true), jeśli 'x' jest całkowita. W przeciwnym przypadku 0 (false)
Arcus tangens hiperboliczny i jego pochodna	double atanh(double x) double dtanh(double x)	atanh(0.1)	Arcus tangens hiperboliczny. Pochodna arcus tangensa hiperbolicznego.
Secans hiperboliczny i jego pochodna	double sech(double x) double dsech(double x)	sech(0.1)	Secans hiperboliczny. Pochodna secansa hiperbolicznego.

Logit	double logit(double x)	logit(0.2)	Funkcja logit: $\frac{1}{1+e^x}$
Efektywność	double effic(double x,double y)	effic(8,11)	Funkcja efektywności: dla nieujemnych argumentów zwracana jest średnia geometryczna, w innych przypadkach 0
Erf – funkcja błędu Gaussa	double erf(double x)	erf(0.0)	Funkcja błędu Gaussa
Φ – gęstość rozkładu normalnego	double phi(double x)	phi(1.5)	Gęstość rozkładu normalnego o średniej 1 i wariancji 1

1.2 Tekst (*text*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Deklaracja zmiennej tekstowej	Text()	Text t	Deklarowana jest zmienna typu tekstowego
Tworzenie tekstu na podstawie ciągu znaków	Text(const char *str)	Text t("jakiś napis")	Tworzony jest tekst zdefiniowany ciągiem znaków
Tworzenie tekstu jednoznakowego	Text(char c)	Text t('a')	Tworzony jest tekst złożony z jednego znaku. W tym przykładzie będzie to "a".
Tworzenie tekstu odpowiadającego liczbie całkowitej	Text(int i)	Text t(2)	Tworzony jest tekst na podstawie liczby całkowitej. W tym przypadku będzie to "2".
Tworzenie tekstu odpowiadającego liczbie rzeczywistej	Text(double d)	Text t(3.95)	Tworzony jest tekst na podstawie liczby rzeczywistej. W tym przypadku będzie to "3.95".
Lorem ipsum	Text lorem_ipsum(int rep=1)	lorem_ipsum()	Funkcja zwraca demonstracyjny tekst „Lorem ipsum". Parametr rep określa liczbę powtórzeń tekstu (domyślnie 1).
Podstawowe konwersje			
Konwersja na ciąg znaków	operator char * ()	(char*)(t)	Tekst jest konwertowany na ciąg znaków
Konwersja na double	operator double()	double(t)	Tekst jest konwertowany n a liczbę typu double

Konwersja na string	string as_string(Text t)	as_string(t)	Tekst jest konwertowany na string.
Dostęp do znaków			
Długość	int len(const Text &t)	len(t)	Zwraca długość tekstu
Znak	char & operator()(int n)	t(5)	Zwraca znak występujący w tekście na pozycji ‘n’.
Operatory			
Operatory porównania	int operator ==(const Text &t1,const Text &t2) int operator <(const Text &t1,const Text &t2) itd.	t1 == t2 t1 < t2 "dwa" == "dwa"	Dwa teksty są porównywane znak po znaku. Dostępne są wszystkie operatory (==,!=,<,>,<=,>=).
Scalanie	Text operator +(const Text &,const Text &)	t1 + t2 "dwa"+"naście"	Dwa teksty są scalane. W wyniku uzyskuje się ciąg znaków pierwszego i drugiego napisu, ustawione jeden za drugim. W drugim przykładzie otrzymamy słowo "dwanaście".
Scalanie przyrostowe	void operator +=(const Text &t)	t1 += t2	Analogicznie jak +, ale przyrostowo.
Zastępowanie znaków			
Zamiana znaku	Text replace(const Text &t,char c1,char c2)	replace("aut",’a’,’b’)	Zwracany jest tekst, w którym znak c1 jest zastąpiony znakiem c2. W tym przypadku zwrócone zostanie słowo „but”.
Zamiana kropek na przecinki	Text dot2com(const Text &t)	dot2com("3.22")	Zwracany jest tekst, w którym kropki (‘.’) są zamienione na przecinki (‘,’). W tym przypadku zwrócone będzie "3,22".
Zamiana przecinków na	Text com2dot(const Text	com2dot("3,22")	Zwracany jest tekst, w którym przecinki (‘,’) są zamienione na kropki (‘.’).

kropki	&t)		W tym przypadku zwrócone będzie "3.22".
Zamiana na wielkie litery	Text to_upper(const Text &t)	to_upper("Kowalski")	Litery w tekście są zamieniane na wielkie. W tym przykładzie wynikiem będzie „KOWALSKI”.
Zamiana na małe litery	Text to_lower(const Text &t)	to_lower("Nowak")	Litery w tekście są zamieniane na małe. W tym przykładzie wynikiem będzie „nowak”.
Usuwanie polskich znaków z tekstu	Text remove_pl(const Text &t)	remove_pl(„rękodzieło”)	Polskie litery w tekście są zamieniane na litery łacińskie. W tym przykładzie wynikiem będzie „rekodzielo”.
Zmiana formatowania znaków z ISO 8859-2 na Windows-1250	Text to_win1250(const Text &)	to_win1250(„tekst”)	Polskie litery zakodowanie zestawem wschodnioeuropiejskim Latin2 są zamieniane na zakodowane zestawem obowiązującym w systemie Windows PL.
Zmiana formatowania znaków z ISO 8859-2 na UTF-8	Text to_utf8(const Text &)	to_utf8(„tekst”)	Polskie litery zakodowanie zestawem wschodnioeuropiejskim Latin2 są zamieniane na zakodowane zestawem UTF-8 (wykorzystywanym m.in. w systemie Linux).
Wybieranie fragmentów			
Fragment początkowy	Text part(const Text &t,int l)	part("dwanaście",3)	Zwraca tekst składający się z ‘l’ początkowych znaków tekstu ‘t’. W tym przypadku wynikiem będzie "dwa".
Fragment	Text subtext(const Text &t,int f,int l)	subtext("dwanaście",2,3)	Zwraca tekst składający się z ‘l’ znaków tekstu ‘t’, zaczynających się od znaku o numerze ‘f’. W tym przypadku wynikiem będzie "wan".
Fragment końcowy	Text endtext(const Text &t,int f)	endtext("dwanaście",4)	Zwraca tekst składający się z końcowych znaków tekstu ‘t’, zaczynający się od znaku o numerze ‘f’. W tym przypadku wynikiem będzie "naście".
Konwersje numeryczne, formatowanie			

Zamiana liczby całkowitej na tekst	Text itot(int i) Text ltot(long l)	Itot(12) ltot(1234567890)	Zwracany jest tekst odpowiadający danej liczbie całkowitej. Funkcja itot konwertuje liczby typu int, a funkcja ltot konwertuje liczby typu long int. W pierwszym przykładzie zwracany jest tekst "12", w drugim „1234567890”.
Zamiana liczby rzeczywistej na tekst	Text dtot(double d,int l=0) Text dtotl(double d) Text dtots(double d)	dtot(3.95) dtot(3.95, 4) dtotl(3.95) dtots(3.95)	Zwracany jest tekst odpowiadający danej liczbie rzeczywistej. Parametr ‘l’ określa liczbę miejsc dziesiętnych, które zostaną zachowane. Dla l=0 konwersja zachowa 6 cyfr po przecinku. Wersja dtotl (long) zachowuje 12 cyfr po przecinku. Wersja dtots (short) zachowuje 2 cyfr po przecinku.
Zamiana liczby całkowitej na tekst i formatowanie w %	Text fperc(double d,int l=2) Text fperc2(double d,int l=2)	fperc(0.12) fperc2(0.12)	Zwracany jest tekst odpowiadający danej liczbie rzeczywistej, sformatowany w procentach z dokładnością do 'l' miejsc po przecinku (domyślnie dwóch) i zakończony znakiem %. Funkcja fperc2 zwraca tekst zakończony dwoma znakami %. W pierwszym przypadku wynikiem jest "12.00%". W drugim "12.00%%".
Zamiana tekstu na liczbę całkowitą	int ttoi(const Text &t)	ttoi("3")	Zwraca liczbę całkowitą odpowiadającą danemu tekstowi.
Zamiana tekstu na liczbę rzeczywistą	double ttod(const Text &t)	ttod("6.5")	Zwraca liczbę rzeczywistą odpowiadającą danemu tekstowi.
Ujęcie w cudzysłów	Text quote(const Text &t)	quote("abc")	Zwraca dany tekst ujęty w cudzysłów. W tym przypadku ""abc""
Sprawdzanie rodzaju			
Czy data	int isdate(const Text &t)	isdate("2012-01-01")	Sprawdzenie czy tekst reprezentuje datę. Zwracane jest 1 (true) lub 0 (false).
Czy liczba	int numerical(const Text &t)	numerical("250.5") numerical("2.505e2")	Sprawdzenie czy tekst reprezentuje liczbę. Zwracane jest 1 (true) lub 0 (false).

Wyszukiwanie fragmentów i zastępowanie			
Sprawdzenie wystąpienia	int exist(char c) int exist(const Text &t)	t.exist(‘a’) t.exist("kot")	Sprawdzenie czy w tekście występuje dana litera lub dany fragment. Zwraca 1 (true) lub 0 (false).
Pierwsza pozycja wystąpienia	int firstpos(char c) int firstpos(const Text &t)	t.firstpos(‘a’) t. firstpos ("kot")	Pierwsza pozycja, na której w tekście występuje dana litera lub dany fragment. Zwraca pozycję lub -1, gdy nie znaleziono.
Ostatnia pozycja wystąpienia	int lastpos(char c) int lastpos(const Text &t)	t.lastpos(‘a’) t. lastpos ("kot")	Ostatnia pozycja, na której w tekście występuje dana litera lub dany fragment. Zwraca pozycję lub -1, gdy nie znaleziono.
Pozycja n-tego wystąpienia znaku	int nthpos(char,int)	t.nthpos('a',6)	Pozycja n-tego wystąpienia znaku w tekście. Zwraca pozycję lub -1, gdy nie znaleziono.
Zastępowanie tekstu	Text replace(const Text &t,const Text &in,const Text &out)	replace(t,"kot","pies")	W tekście t ciąg znaków wejściowy ‘in’ jest zastępowany ciągiem wyjściowym ‘out’. Gdy ‘in’ występuje więcej niż raz, to zastępowanie dotyczy wszystkich wystąpień. W przykładzie słowo ‘kot’ jest zastępowane słowem ‘pies’.
Operacje wejścia/wyjścia			
Wyświetlenie	ostream & operator <<(ostream &out,const Text &t)	cout<<t	Wyświetlenie tekstu
Pobranie	istream & operator >>(istream &in,Text &t)	cin>>t	Pobranie tekstu wpisywanego z klawiatury
Odczyt / zapis do pliku	void read(FILE *file) void write(FILE *file)	t.read(file) t.write(file)	Odczyt/zapis tekstu do pliku typu FILE.
Wczytanie treści z pliku	void read_file(const char	t.read_file(„dane.txt”)	Wczytanie do zmiennej tekstowej treści z pliku o danej nazwie.

o danej nazwie	*name)		
Wczytanie treści z pliku do stringa	string read_file(const char *name)	read_file(„dane.txt”)	Wczytanie do stringa treści z pliku o danej nazwie.

1.3 Data (*date*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Deklaracja daty	date()	date da	Deklarowana jest zmienna typu data
Tworzenie daty na podstawie napisu	date(const char *str) date(const Text t)	date da("2014-12-06")	Tworzona jest zmienna data na podstawie napisu
Tworzenie daty na podstawie składowych rok, miesiąc i dzień	date(long y,long m,long d)	date da(2014,12,6)	Tworzona jest zmienna data na podstawie składowych rok, miesiąc i dzień
Tworzenie daty			
Zamiana napisu w datę	date d(const char *str)	date da = d("2014-12-06")	Zwraca datę utworzoną na podstawie napisu
Dzisiaj	date today()	date da = today()	Zwraca bieżącą datę
Pierwszy dzień miesiąca	date ymfirst(const date &t)	ymfirst(d("2014-12-06"))	Zwraca pierwszy dzień miesiąca, w którym występuje data ‘t’. W tym przypadku 2014-12-1
Ostatni dzień miesiąca	date ymlast(const date &t)	ymlast(d("2014-12-06"))	Zwraca ostatni dzień miesiąca, w którym występuje data ‘t’. W tym przypadku 2014-12-31
Konwersja			
Konwersja daty na liczbę	operator double()	double(d("2014-12-06"))	Konwertuje datę na liczbę rzeczywistą. Zachowywany jest rok (jako części całkowite) i miesiąc (na miejscach dziesiętnych). Dzień miesiąca jest ignorowany. W tym przykładzie zwracana jest liczba 2014.12.

Konwersja liczby na datę	date dtoda(double d)	date da=dtoda(2014.12)	Liczba rzeczywista jest zamieniana na datę. Liczba rzeczywista musi zawierać rok (jako części całkowite) i miesiąc (na miejscach dziesiętnych). W przykładzie liczba 2014.12 jest zamieniana na datę 2014-12-1
Konwersja daty na tekst	Text datot(const date &da)	datot(d("2014-12-06"))	Konwertuje datę na tekst. W tym przykładzie zwracany jest tekst "2014-12-06".
Składowe daty			
Dzień	long day(const date &da)	day(d("2014-12-06"))	Zwraca dzień dla podanej daty. W tym przypadku 6
Miesiąc	long month(const date &da)	month(d("2014-12-06"))	Zwraca miesiąc dla podanej daty. W tym przypadku 12
Rok	long year(const date &da)	year(d("2014-12-06"))	Zwraca rok dla podanej daty. W tym przypadku 2014
Składowe rok, miesiąc, dzień	void ymd(long &y,long &m,long &d)	da.ymd(y,m,d)	Rozkłada datę na składowe rok (y), miesiąc (m) i dzień (d).
Przesuwanie dat			
Zwiększenie o liczbę dni	date operator +(const date &da,const long l)	d("2014-12-06")+40	Zwiększa datę o 'l' dni. W tym przypadku wynikiem jest 2015-01-15
Zmniejszenie o liczbę dni	date operator -(const date &da,const long l)	d("2014-12-06")-40	Zmniejsza datę o 'l' dni. W tym przypadku wynikiem jest 2014-10-27
Modyfikacja o liczbę dni	date dmod(const date &da,const long l)	dmod(d("2014-12-06"),-40)	Modyfikuje datę o 'l' dni. 'l' dodatnie powoduje zwiększenie, a 'l' ujemne zmniejszenie daty. W tym przypadku wynikiem jest 2014-10-27

Modyfikacja o liczbę miesięcy	date mmod(const date &da,const long l)	mmod(d("2014-12-06"),2)	Modyfikuje datę o 'l' miesięcy. 'l' dodatnie powoduje zwiększenie, a 'l' ujemne zmniejszenie daty. Jeśli data 'da' jest z końca miesiąca (np. 31), to po przesunięciu dzień jest zmiejszany (np. na 30), po to, by data wynikowa była poprawną datą. W tym przypadku wynikiem jest 2015-2-6
Modyfikacja o liczbę lat	date ymod(const date &da,const long l)	ymod(d("2014-12-06"),-2)	Modyfikuje datę o 'l' lat. 'l' dodatnie powoduje zwiększenie, a 'l' ujemne zmniejszenie daty. W tym przypadku wynikiem jest 2012-12-6
Różnica i porównywanie dat			
Różnica	long operator -(const date &da1,const date &da2)	d("2014-12-26")-d("2012-12-06")	Obliczana jest różnica między datami (wyrażona w dniach). W tym przypadku różnica wynosi 750 dni.
Porównywanie	int operator ==(const date &da1,const date &da2) int operator <(const date &da1,const date &da2) itd.	d("2014-12-26") == d("2012-12-06") d("2014-12-26") < d("2012-12-06")	Daty są porównywane z dokładnością do dnia. Dostępne są wszystkie operatory (==,!=,<,>,<=,>=).
Obsługa strumieni			
Wyświetlenie	ostream & operator <<(ostream &out,const date &da)	cout<<d("2014-12-06")	Wyświetlenie daty
Pobranie	istream & operator >>(istream &in,date &da)	cin>>da	Pobranie daty wpisywanej z klawiatury. Oczekiwany jest format yyyy-mm-dd. Separatorem może też być kropka lub /.

1.4 Wektor (*vector*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Utworzenie wektora	vector()	vector v	Tworzony jest pusty wektor v
Utworzenie wektora danej długości	vector(int n_new)	vector v(5)	Tworzony jest wektor v o długości 5
Dostęp do elementów			
Długość wektora	int length(const vector &v)	length(v)	Zwracana jest długość wektora
Element wektora	int & operator ()(int i)	v(3)	Dostęp do 3 elementu wektora v
Tworzenie wektorów			
Wektor stałych	vector c(int c1,int c2,int c3,int c4,int c5,int c6,int c7,int c8,int c9,int c10)	c(2,4,8)	Powstaje wektor o wartościach podanych jako argumenty. W ten sposób można tworzyć wektory o długości do 10. Dłuższe wektory uzyskuje się funkcją 'vi'. W tym przypadku powstaje wektor o elementach [2,4,8].
Wektor stałych danej długości	vector vi(int l,...)	vi(3,2,4,8)	Powstaje wektor o długości 'l' i wartościach podanych jako kolejne argumenty. W tym przypadku wektor o długości 3 i elementach [2,4,8]
Sekwencja liczb od 1	vector $(int b)	$(10)	Powstaje sekwencja liczb od 1 do b. W tym przypadku [1,2,3,4,5,6,7,8,9,10]
Sekwencja liczb	vector $(int a,int b)	$(3,8)	Powstaje sekwencja liczb od a do b. W tym przypadku [3,4,5,6,7,8]
Sekwencja ze skokiem	vector $(int a,int step,int b)	$(1,2,9)	Powstaje sekwencja liczb od a do b, krokiem ‘step’. W tym przypadku [1,3,5,7,9]
Wypełnianie stałą	void fill(int c)	v.fill(8)	Wypełnianie wektora stałą. W tym przypadku wektor v zostaje wypełniony wartościami 8

Wektor stałych	vector filled(int c,int l)	filled(8,5)	Powstaje wektor o długości ‘l’, wypełniony stałą ‘c’. W tym przypadku [8,8,8,8,8]
Wektor zer	vector vzeros(int l)	vzeros(5)	Powstaje wektor o długości ‘l’, wypełniony zerami. W tym przypadku [0,0,0,0,0]
Wektor jedynek	vector vones(int l)	vones(5)	Powstaje wektor o długości ‘l’, wypełniony jedynkami. W tym przypadku [1,1,1,1,1]
Funkcje specjalne			
Suma wartości	int sum(const vector &v)	sum(v)	Suma elementów wektora
Wyszukanie pozycji niezerowych	vector find(const vector &v)	find(v)	Wyszukuje pozycje, na których w wektorze są wartości różne od 0. Wynikiem jest wektor indeksów, np. v=c(2,0,8) v2=find(v) v to wektor [2,0,8]. Wówczas v2 to wektor [1,3]
Indeks przeciwny do danego indeksu	vector nindex(const vector &v,int nl)	nindex(v,15)	Tworzy wektor indeksów przeciwnych do danego, z zakresu od 1 do nl. Np. v=c(-1,-3,-5,-7,-9) v2=nindex(v,15) v to wektor [-1,-3,-5,-7,-9]. Wówczas v2 to wektor [2,4,6,8,10,11,12,13,14,15]
Zmiana długości	void resize(int n_new)	v.resize(10)	Zmiana długości wektora.
Zwolnienie pamięci	void free()	v.free()	Zwolnienie pamięci przydzielonej dla wektora.
Operacja zmiany znaku	Vector operator -(const Vector &)	-v	Zmiana znaku elementów wektora na przeciwny.
Operatory logiczne			
Zaprzeczenie	vector operator !(const vector &v)	!v	Zaprzeczenie elementów wektora. Zera zamieniane są na jedynki, inne wartości na zera.

Koniunkcja	vector operator &&(const vector &v1,const vector&v2)	v1&&v2	Koniunkcja po elementach wektorów v1 i v2
Alternatywa	vector operator ||(const vector &a,const vector &b)	v1||v2	Alternatywa po elementach wektorów v1 i v2
Wyświetlanie			
Wyświetlenie wektora	void display(Text name="")	v.display()	Wyświetlanie wektora. Jako parametr można podać nazwę do wyświetlenia.

1.5 Dane (*data*)

Wszystkie funkcje i metody wymienione w tym rozdziale działają zarówno dla obiektów typu macierz (*matrix*) jak i tabela (*table*).

1.5.1 Deklaracja zmiennej

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Utworzenie pustych danych	data()	data d	Tworzone są puste dane
Utworzenie danych o zadanych wymiarach	data(int m,int m)	data d(5,3)	Tworzone są dane o wymiarach m x n (m wierszy, n kolumn). W tym przypadku tworzone są dane o wymiarach 5x3 (5 wierszy, 3 kolumny).
Utworzenie danych o zadanym wymiarze	data(int m)	data d(3)	Tworzone są dane wymiarach m x m. W tym przypadku tworzone są dane o wymiarach 3x3.

1.5.2 Wymiary obiektu

Wymiary obiektu			
Liczba wierszy	int size1(const data &a) int nrow(const data &a)	size1(d) nrow(d)	Zwraca liczbę wierszy obiektu
Liczba kolumn	int size2(const data &a)	size2(d)	Zwraca liczbę kolumn obiektu

	int ncol(const data &a)	ncol(d)	
Długość	int length(const data &a)	length(d)	Zwraca długość obiektu (większy z wymiarów). W przypadku obiektów, których jeden z wymiarów wynosi 1, funkcja ta ma naturalną interpretację.

1.5.3 Dostęp do obiektu

Dostęp do etykiet			
Etykieta wiersza o danym numerze	Text & rowname(int i)	d.rowname(1)	Zwraca etykietę wiersza o numerze 'i'. W tym przypadku pierwszego
Etykieta kolumny o danym numerze	Text & colname(int j)	d.colname(5)	Zwraca etykietę kolumny o numerze 'j'. W tym przypadku 5
Dostęp do elementów			
Dostęp do elementu za pomocą numeru wiersza i numeru kolumny	T & operator ()(int i,int j)	d(2,3)	Zwraca element z wiersza ‘I’ i kolumny ‘j’. W tym przypadku z wiersza 2 i kolumny 3
Dostęp do elementu za pomocą numeru wiersza i nazwy kolumny	T & operator ()(int i,Text name2)	d(2,”COL3”)	Zwraca element z wiersza ‘I’ i kolumny o nazwie name2. W tym przypadku z wiersza 2 i kolumny o nazwie „COL3”
Dostęp do elementu za pomocą nazwy wiersza i numeru kolumny	T & operator ()(Text name1,int j)	d(“ROW2”,3)	Zwraca element z wiersza o nazwie name1 i kolumny ‘j’. W tym przypadku z wiersza o nazwie „ROW2” i kolumny 3
Dostęp do elementu za pomocą nazwy wiersza	T & operator ()(Text name1,Text name2)	d(“ROW2”,”COL3”)	Zwraca element z wiersza o nazwie name1 i kolumny o nazwie name2. W tym przypadku z wiersza o nazwie „ROW2” i kolumny o nazwie „COL3”

i nazwy kolumny			
Dostęp do elementu wektora	T & operator ()(int i)	d(5)	Jeśli obiekt na jeden z wymiarów równy 1, czyli jest wektorem, aby dostać się do jego elementów wystarczy podać tylko jeden argument – pozycję elementu. W tym przypadku zwracany jest element piąty.
Dostęp do kolumny			
Wybór kolumny na podstawie jej nazwy	data operator [](Text name2)	d[“WIEK”]	Zwracana jest kolumna danych o nazwie name2. W tym przypadku kolumna o etykiecie „WIEK”
Wybór kolumny na podstawie jej numeru	data operator [](int j)	d[16]	Zwracana jest kolumna danych o numerze 'j'. W tym przypadku kolumna 16

1.5.4 Wyszukiwanie

Wyszukiwanie wiersza, kolumny			
Wyszukiwanie numeru wiersza	int find_row(Text name1,int upper=1)	d.find_row(„ROW1”)	Zwraca numer wiersza o etykiecie name1. W przypadku nieznalezienia, zwraca 0. Parametr upper=1 oznacza, że wielkość liter w nazwie nie będzie brana pod uwagę. W tym przypadku wyszukiwany jest wiersz o etykiecie „ROW1”.
Wyszukiwanie numeru kolumny	int find_col(Text name2,int upper=1)	d.find_col(„COL5”)	Zwraca numer kolumny o etykiecie name2. W przypadku nieznalezienia, zwraca 0. Parametr upper=1 oznacza, że wielkość liter w nazwie nie będzie brana pod uwagę. W tym przypadku wyszukiwana jest kolumna o etykiecie „COL5”.
Wyszukiwanie elementu			
Wyszukiwanie elementu w kolumnie o danym numerze	int find_cell(T cell,int col,int row0=1)	d.find_cell(“10 lat”,16)	Przeszukuje kolumnę o numerze col, żeby znaleźć element cell. Zwraca numer wiersza, w którym jest element cell. Jeśli nie znaleziono szukanego elementu, funkcja zwraca 0. Domyślnie szukanie rozpoczyna się od wiersza o numerze 1. Można wybrać inny wiersz startowy (parametr row0). W tym przypadku szukany jest element „10 lat” w kolumnie 16, począwszy od wiersza 1.

Wyszukiwanie elementu w kolumnie o danej nazwie	int find_cell(T cell,Text cname,int row0=1,int upper=1)	d.find_cell("10 lat","WIEK",1000)	Przeszukuje kolumnę o nazwie cname, żeby znaleźć element cell. Zwraca numer wiersza, w którym jest element cell. Jeśli nie znaleziono szukanego elementu, funkcja zwraca 0. Domyślnie szukanie rozpoczyna się od wiersza o numerze 1. Można wybrać inny wiersz startowy (parametr row0). Parametr upper=1 oznacza, że wielkość liter w nazwie nie będzie brana pod uwagę. W tym przypadku szukany jest element „10 lat" w kolumnie o nazwie „WIEK", począwszy od wiersza 1000.

1.5.5 Zmiana rozmiaru i kształtu

Zmiana rozmiaru i kształtu			
Zmiana rozmiaru obiektu	void resize(int m_new,int n_new)	d.resize(5,10)	Zmiana wymiarów obiektu. Po zmianie obiekt ma m_new wierszy i n_new kolumn. Zawartość obiektu ulega skasowaniu. W tym przypadku obiekt będzie miał wymiary 5x10
Zmiana rozmiaru obiektu z zachowaniem zawartości	void resizec(int m_new,int n_new)	d.resizec(5,10)	Zmiana wymiarów obiektu. Po zmianie obiekt ma m_new wierszy i n_new kolumn. Zawartość obiektu jest zachowywana. Jeśli nowe wymiary są większe od pierwotnych, to elementy nie występujące w obiekcie pierwotnym pozostają puste (dla typu tekst) lub 0 (dla typu liczbowego). W tym przypadku obiekt będzie miał wymiary 5x10
Przegrupowanie obiektu	void reshape(int m_new,int n_new)	d.reshape(5,10)	Przegrupowanie obiektu pozwala zmienić układ elementów obiektu. Wymagane jest, aby liczba elementów pierwotnego i docelowego była taka sama. Elementy są kopiowane kolumnami. W tym przypadku jeśli 'd' ma 50 elementów (np. ma wymiary 25x2), to może zostać przegrupowane do obiektu o wymiarach 5x10.
Zmiana w wektor	data vect(const data &a)	vect(d)	Obiekt 'a' jest przegrupowywany w wektor kolumnowy. Elementy są kopiowane kolumnami.
Dodanie nowych wierszy	void newrows(int m2)	d.newrows(100)	Dodawane są nowe wiersze do danych. Liczba nowych wierszy wynosi m2. W tym przypadku dochodzi 100 wierszy.

Dodanie nowej kolumny	void newcol(Text name2)	d.newcol(„DOCHOD”)	Dodawana jest nowa kolumna o nazwie name2 do danych. W tym przypadku kolumna o etykiecie „DOCHOD”

1.5.6 Sprawdzanie zawartości

Sprawdzanie zawartości			
Czy pusty	int isempty(const data &a)	isempty(d)	Sprawdzenie czy obiekt 'a' jest pusty. Zwraca 0 (false) lub 1 (true)
Czy kwadratowy	int issqr(const data &a)	issqr(d)	Sprawdzenie czy obiekt 'a' jest kwadratowy. Zwraca 0 (false) lub 1 (true)
Czy symetryczny	int issym(const data &a)	issym (d)	Sprawdzenie czy obiekt 'a' jest symetryczny względem głównej przekątnej. Zwraca 0 (false) lub 1 (true)
Czy jakikolwiek istotny	int any(const data &a)	any(d)	Sprawdzenie czy obiekt 'a' zawiera jakikolwiek istotny element. W przypadku elementów tekstowych nieistotny jest ciąg pusty znaków („”). W przypadku elementów liczbowych nieistotne są zera. Zwraca 0 (false) lub 1 (true)
Czy wszystkie istotne	int all(const data &a)	all(d)	Sprawdzenie czy wszystkie elementy obiektu 'a' są istotne. W przypadku elementów tekstowych nieistotny jest ciąg pusty znaków („”). W przypadku elementów liczbowych nieistotne są zera. Zwraca 0 (false) lub 1 (true)

1.5.7 Tworzenie wektorów

Wektory stałych			
Utworzenie wektora	data c(T,T,T,T,T,T,T,T,T,T)	c(10,20,30,40,50) c(“Ala”,”ma”,”kota”)	Tworzony jest wektor na podstawie stałych. W ten sposób można tworzyć wektory o długości do 10. Dłuższe należy tworzyć za pomocą funkcji ‘vd’ dostępnej dla macierzy.

			W pierwszym przypadku wektor liczb [10,20,30,40,50]. W drugim wektor tekstowy [„Ala”,”ma”,”kota”].
Powielanie			
Powielenie zadaną liczbę razy	data rep(const data &v,int n)	rep(d,2)	Zwraca dany wektor v powielony n razy. W tym przypadku 2 razy
Powielenie do danej długości	data replen(const data &v,int l)	replen(d,10)	Zwraca dany wektor v powielony tyle razy, aby wektor wynikowy miał długość l. W tym przypadku zwracany jest wektor o długości 10
Powielenie każdego elementu daną liczbę razy	data repeach(const data &v,int n)	repeach(d,2)	Zwraca wektor v składający się z elementów wektora d, każdego powtórzonego n razy. W tym przypadku zwracany jest wektor składający się z elementów wektora d, każdy element powtórzony 2 razy.
Powielenie każdego elementu zdefiniowaną liczbę razy	data rep(const data &v,const vector &index)	rep(d,indeks)	Zwraca wektor składający się z elementów wektora v, każdego powtórzonego zdefiniowaną w wektorze index liczbę razy. Przykładowo: d to wektor [„Kot”, „Pies”] indeks to wektor [2,3] Wówczas zwracany jest wektor [„Kot”, ”Kot”, ”Pies” ,”Pies”, ”Pies”]
Utworzenie danych na podstawie dwóch wektorów i funkcji	data outer(const data &v1,const data &v2,T fun(T,T))	outer(d1,d2,suma)	Tworzony jest obiekt prostokątny, którego elementy są wynikiem funkcji ‘fun’ wywołanej z argumentami pochodzącymi z wektora v1 i v2. Elementy obiektu wyjściowego o współrzędnych (i,j) przyjmują wartości fun(v1(i),v2(j)). W tym przypadku zwracany jest obiekt, którego elementy to wywołanie funkcji ‘suma’ dla odpowiednich elementów d1 i d2.

1.5.8 Zmiana zawartości

Wypełnianie			
Wypełnianie stałą	void fill(T t)	d.fill(„ABC”)	Obiekt jest wypełniany stałą wartością ‘t’. W tym przypadku, wszystkie elementy są wypełnione tekstem „ABC”
Wypełnianie stałą w zakresie	void fill(T t,int m0,int n0,int m_num,int n_num)	d.fill(„ABC”,1,2,10,20)	Obiekt jest wypełniany stałą wartością ‘t’ w zadanym zakresie. W zakresie od wiersza ‘m0’ i kolumny ‘n0’ wypełnianych jest ‘m_num’ wierszy i ‘n_num’ kolumn. W tym przypadku następuje wypełnienie tekstem „ABC” dziesięciu wierszy zaczynając od pierwszego i 20 kolumn zaczynając od drugiej.
Czyszczenie	void clear(int clnames=0)	d.clear()	Obiekt jest czyszczony. Jeśli elementy są typu tekst, wstawiany jest pusty tekst („”). Jeśli elementy są liczbowe, wstawiane są zera. Można wybrać, czy usunąć etykiety kolumn i wierszy (parametr ‘clnames’)
Aplikowanie funkcji			
Aplikowanie funkcji do elementów	void forall(T f(T))	d.forall(f)	Na elementy obiektu nakładana jest funkcja ‘f’
Aplikowanie funkcji do danej kolumny	void forcol(int n,T f(T))	d.forcol(2,f)	Na elementy obiektu w kolumnie ‘n’ nakładana jest funkcja ‘f’
Aplikowanie funkcji do danego wiersza	void forrow(int n,T f(T))	d.forrow(3,f)	Na elementy obiektu w wierszu ‘n’ nakładana jest funkcja ‘f’
Tworzenie obiektu poprzez aplikowanie funkcji do elementów	data forall(const data &d,T f(T))	forall(d,f)	Tworzony jest obiekt, którego element powstają z elementów ‘d’ poprzez nałożenie funkcji ‘f’.
Tworzenie obiektu poprzez aplikowanie funkcji do danej	data forcol(const data &d,int n,T f(T))	forcol(d,2,f)	Tworzony jest obiekt, którego element powstają z elementów ‘d’. Na kolumnę o numerze ‘n’ nakładana jest funkcja ‘f’.

kolumny			
Tworzenie obiektu poprzez aplikowanie funkcji do danego wiersza	data forrow(const data &d,int n,T f(T))	forrow(d,3,f)	Tworzony jest obiekt, którego element powstają z elementów ‘d’. Na wiersz o numerze ‘n’ nakładana jest funkcja ‘f’.

1.5.9 Kopiowanie

Kopiowanie zawartości			
Kopiowanie fragmentu (6 parametrów zakresu)	void copy(const data &src,int m0_src,int n0_src,int m_num,int n_num,data &dst,int m0_dst,int n0_dst)	copy(d,1,1,10,20,e,3,2)	Kopiowanie fragmentu obiektu ‘src’ do obiektu ‘dst’. Można wybrać wiersz początkowy i liczbę wierszy (parametry ‘m0_src’,’m_num’) oraz kolumnę początkową i liczbę kolumn do skopiowania (parametry ‘n0_src’, ‘n_num’). Można też określić, w które miejsce obiektu ‘dst’ wkopiować (parametry ‘m0_dst’, ‘n0_dst’). Operacja kopiowania jest zabezpieczona przed przekroczeniem zakresów obiektu docelowego. W tym przypadku kopiuje się z ‘d’ do ‘e’. Pierwszych 10 wierszy i pierwszych 20 kolumn jest kopiowanych w miejsce zaczynające się od 3 wiersza i drugiej kolumny ‘e’.
Kopiowanie fragmentu (4 parametry zakresu)	void copy(const data &src,int m0,int n0,int m_num,int n_num,data &dst)	copy(d,1,1,10,20,e)	Kopiowanie fragmentu obiektu ‘src’ do obiektu ‘dst’. Można wybrać wiersz początkowy i liczbę wierszy (parametry ‘m0’,’m_num’) oraz kolumnę początkową i liczbę kolumn do skopiowania (parametry ‘n0’, ‘n_num’). Operacja kopiowania jest zabezpieczona przed przekroczeniem zakresów obiektu docelowego. W tym przypadku kopiuje się z ‘d’ do ‘e’. Pierwszych 10 wierszy i pierwszych 20 kolumn jest kopiowanych w miejsce zaczynające się od pierwszego wiersza i pierwszej kolumny ‘e’.
Kopiowanie fragmentu (2 parametry zakresu)	void copy(const data &src,int m_num,int n_num,data &dst)	copy(d,10,20,e)	Kopiowanie fragmentu obiektu ‘src’ do obiektu ‘dst’. Można wybrać liczbę wierszy (parametry ’m_num’) oraz liczbę kolumn do skopiowania (parametry ‘n_num’). Operacja kopiowania jest zabezpieczona przed przekroczeniem zakresów obiektu docelowego. W tym przypadku kopiuje się z ‘d’ do ‘e’. Pierwszych 10 wierszy i pierwszych 20 kolumn jest kopiowanych w miejsce zaczynające się od pierwszego wiersza i pierwszej kolumny ‘e’.

Kopiowanie fragmentu (bez parametrów zakresu)	void copy(const data &src,data &dst)	copy(d,e)	Kopiowanie fragmentu obiektu ‘src’ do obiektu ‘dst’. Operacja kopiowania jest zabezpieczona przed przekroczeniem zakresów obiektu docelowego. W tym przypadku kopiuje się z ‘d’ do ‘e’.
Kopiowanie etykiet			
Kopiowanie etykiet z wierszy do wierszy	void copynames_r2r(const data &src,data &dst)	copynames_r2r(d,e)	Kopiowane są etykiety wierszy z obiektu ‘src’ do etykiet wierszy ‘dst’.
Kopiowanie etykiet z kolumn do kolumn	void copynames_c2c(const data &src,data &dst)	copynames_c2c(d,e)	Kopiowane są etykiety kolumn z obiektu ‘src’ do etykiet kolumn ‘dst’.
Kopiowanie etykiet z kolumn do wierszy	void copynames_c2r(const data &src,data &dst)	copynames_c2r(d,e)	Kopiowane są etykiety kolumn z obiektu ‘src’ do etykiet wierszy ‘dst’.
Kopiowanie etykiet z wierszy do kolumn	void copynames_r2c(const data &src,data &dst)	copynames_r2c(d,e)	Kopiowane są etykiety wierszy z obiektu ‘src’ do etykiet kolumn ‘dst’.
Kopiowanie etykiet	void copynames(const data &src,data &dst)	copynames(d,e)	Kopiowane są etykiety wierszy i kolumn z obiektu ‘src’ do ‘dst’.

1.5.10 Wybieranie i wstawianie

Wybieranie i wstawianie kolumn			
Wybór kolumn/kolumny	data col(const data &d,const vector &index) data col(const data &d,int	col(d,c(1,2,3)) col(d,5)	Z obiektu ‘d’ wybierane są kolumny zdefiniowane przez ‘index’. Można wskazać jedną kolumnę. W pierwszym przypadku wybierane są kolumny 1, 2 i 3.

	no) data col(const vector &index) data col(int no)	d.col(c(1,2,3)) d.col(5)	W drugim przypadku wybierana jest kolumna piąta.
Wstawienie kolumn/kolumny	setcol(const vector &index,const data &cols) setcol(int no,const data &cols)	d.setcol(c(1,2,3),cols) d.setcol(5,col7)	Do obiektu ‘d’ wstawiane są kolumny ‘cols’ w miejsce zdefiniowane przez wektor ‘index’. Można wstawić jedną kolumnę, wskazując numer. W pierwszym przypadku do ‘d’ wstawiane są ‘cols’ do 1, 2 i 3 kolumny. W drugim przypadku do piątej kolumny ‘d’ wstawiana jest kolumna ‘col7’
Wybieranie i wstawianie wierszy			
Wybór wiersza/wierszy	data row(const data &d,const vector &index) data row(const data &d,int no) data row(const vector &index) data row(int no)	row(d,c(1,2,3)) row(d,5) d.row(c(1,2,3)) d.row(5)	Z obiektu ‘d’ wybierane są wiersze zdefiniowane przez ‘index’. Można wskazać jeden wiersz. W pierwszym przypadku wybierane są wiersze 1, 2 i 3. W drugim przypadku wybierany jest wiersz piąty.
Wstawienie wiersza/wierszy	setrow(const vector &index,const data &rows) setrow(int no,const data &rows)	d.setrow(c(1,2,3),rows) d.setrow(5,row7)	Do obiektu ‘d’ wstawiane są wiersze ‘rows’ w miejsce zdefiniowane przez wektor ‘index’. Można wstawić jeden wiersz, wskazując numer. W pierwszym przypadku do ‘d’ wstawiane są ‘rows’ do 1, 2 i 3 wiersza. W drugim przypadku do piątego wiersza ‘d’ wstawiany jest wiersz ‘row7’
Wybieranie i wstawianie			
Wybór wierszy i kolumn	data get(const data &src,const vector	get(d,c(1,2),c(3,5))	Z obiektu ‘src’ wybierane są wiersze i kolumny zdefiniowane indeksami.

	&index_rows,const vector &index_cols) data get(const vector &index_rows,const vector &index_cols)	d.get(c(1,2),c(3,5))	W tym przypadku z ‘d’ wybierane są elementy z wierszy 1 i 2 oraz kolumn 3 i 5.
Wstawianie wierszy i kolumn	void set(const vector &index_rows_dst,const vector &index_cols_dst,const data &src,const vector &index_rows_src,const vector &index_cols_src)	e.set(c(20,15),c(1,2),d, c(1,2),c(3,5))	Do obiektu w miejsce zdefiniowane indeksami docelowymi wstawiany jest fragment obiekt ‘src’ zdefiniowany indeksami źródłowymi. W tym przypadku z ‘d’ wybierane są elementy z wierszy 1 i 2 i kolumn 3 i 5. Są one wstawiane do ‘e’ do wierszy 20 i 15 i kolumn 1 i 2.
Wstawianie wierszy i kolumn (bez definicji indeksów źródłowych)	void set(const vector & index_rows_dst,const vector & index_cols_dst,const data &src)	e.set(c(20,15),c(1,2),d)	Do obiektu w miejsce zdefiniowane indeksami docelowymi wstawiany jest obiekt ‘src’. W tym przypadku z obiekt ‘d’ jest wstawiany do ‘e’ do wierszy 20 i 15 i kolumn 1 i 2.

1.5.11 Łączenie

Łączenie			
Łączenie obiektów poprzez dodanie kolumn	data operator | (const data &a,const data &b)	a|b	Obiekty ‘a’ i ‘b’ są łączone poprzez ich zestawienie obok siebie. Wymagana jest zgodność liczby wierszy.
Dołączanie kolumn	void operator |=(const data &a)	a|=b	Obok obiektu ‘a’ jest dostawiany ‘b’. Wymagana jest zgodność liczby wierszy.

Łączenie obiektów poprzez dodanie wierszy	data operator & (const data &a,const data &b)	a&b	Obiekty ‘a’ i ‘b’ są łączone poprzez ich zestawienie jeden nad drugim. Wymagana jest zgodność liczby kolumn.
Dołączanie wierszy	void operator &=(const data &a)	a&=b	Pod obiektem ‘a’ jest dostawiany ‘b’. Wymagana jest zgodność liczby kolumn.
Tasowanie kolumn	data shufflecol(const data &a,const data &b)	shufflecol(a,b)	Między kolumny ‘a’ są równomiernie wstawiane kolumny ‘b’
Tasowanie wierszy	data shufflerow(const data &a,const data &b)	shufflerow(a,b)	Między wiersze ‘a’ są równomiernie wstawiane wiersze ‘b’

1.5.12 Operacje zmiany orientacji

Operacje zmiany orientacji			
Transpozycja	data operator ~ (const data &a) void tr()	~d d.tr()	Transpozycja wierszy i kolumn. W wersji ‘tr()’ obiekt musi być kwadratowy.
Zamiana lewo-prawo	data fliplr(const data &a) void fliplr()	fliplr(d) d.fliplr()	Zamiana elementów z lewej na prawo.
Zamiana góra-dół	data flipud(const data &a) void flipud()	flipud(d) d.flipud()	Zamiana elementów z gór na dół.
Obrót o 90 stopni	data rot90(const data &a) void rot90()	rot90(d) d.rot90()	Obrót o 90 stopni.

Przesunięcie w górę	data up(const data &a) void up()	up(d) d.up()	Przesunięcie trójkątne w górę.
Przesunięcie w dół	data down(const data &a) void down()	down(d) d.down()	Przesunięcie trójkątne w dół.
Dolny trójkąt	data tril(const data &a) void tril()	tril(d) d.tril()	Dolny trójkąt (górny wypełniony „” lub 0).
Górny trójkąt	data triu(const data &a) void triu()	triu(d) d.triu()	Górny trójkąt (dolny wypełniony „” lub 0).

1.5.13 Zamiana wierszy, kolumn

Zamiana wierszy, kolumn			
Zamiana wierszy	void swap_rows(int r1,int r2)	d.swap_rows(2,10)	Zamiana wierszy. W tym przypadku zamiana wiersza 2 z 10.
Zamiana kolumn	void swap_cols(int c1,int c2)	d.swap_cols(2,10)	Zamiana kolumn. W tym przypadku zamiana kolumny 2 z 10.
Losowy układ wierszy	void random_shuffle_rows()	d.random_shuffle_rows()	Wiersze są układane w losowej kolejności
Losowy układ kolumn	void random_shuffle_cols()	d.random_shuffle_cols()	Kolumny są układane w losowej kolejności

Zmiana kolejności wierszy	void reorder_rows(const vector &ord,int reord_values=1,int reord_labels=1)	d.reorder_rows(ord)	Wiersze są układane w kolejności zdefiniowanej przez wektor ord. Można określić, czy zmiana kolejności dotyczy wartości (parametr reord_values), czy etykiet (parametr reord_labels). Domyślnie zamieniane są i wartości, i etykiety.
Zmiana kolejności kolumn	void reorder_cols(const vector &ord,int reord_values=1,int reord_labels=1)	d.reorder_cols(ord)	Kolumny są układane w kolejności zdefiniowanej przez wektor ord. Można określić, czy zmiana kolejności dotyczy wartości (parametr reord_values), czy etykiet (parametr reord_labels). Domyślnie zmieniane są i wartości , i etykiety.

1.5.14 Pozostawianie wybranych wierszy, kolumn

Pozostawianie wybranych wierszy, kolumn			
Pozostawianie wierszy od-do	void leave_rows(int r1,int r2)	d.leave_rows(50,100)	Pozostawianie wierszy od r1 do r2
Pozostawianie kolumn od-do	void leave_cols(int c1,int c2)	d.leave_cols(50,100)	Pozostawianie kolumn od c1 do c2
Pozostawianie wierszy do danego	void leave_rows_to(int r2)	d.leave_rows_to(100)	Pozostawianie wierszy od 1 do r2
Próbka losowa określona ilościowo	data sample_n(int num,int repl=0)	d.sample_n(1000)	Losuje próbkę o liczności num z danych. Parametr repl określa, czy losowanie ma być ze zwracaniem (domyślnie bez zwracania). W przykładzie wylosowanych będzie 1000 przypadków bez zwracania.
Próbka losowa określona udziałem	data sample_frac(double frac,int repl=0)	d.sample_frac(0.5,1)	Losuje próbkę o zdefiniowanym udziale z danych. Parametr frac określa udział. Parametr repl określa, czy losowanie ma być ze zwracaniem (domyślnie bez zwracania). W przykładzie wylosowanych będzie 50% przypadków ze zwracaniem.

Operacje warunkowe			
Filtrowanie danych	data filter(const data &d,const vector &v) data filter(const vector &v)	filter(d,v) d.filter(v)	Z obiektu ‘d’ są wybierane wiersze, dla których wektor warunkowy ‘v’ przyjmuje wartości 1 (pomijane są te z 0). Długość ‘v’ musi być zgodna z liczbą wierszy ‘d’. Druga wersja funkcji odwołuje się do obiektu ‘d’.
Dopasowywanie wektorów	vector match(const data &v1,const data &v2,int fromlast=0,int nomatch=0)	match(d,e)	Sprawdzanie, na których pozycjach ‘v2’ są elementy ‘v1’. Zwracany jest wektor pozycji. Można wybrać kolejność działania funkcji (parametr ‘fromlast’) i co wstawić w przypadku, gdy nie znaleziono dopasowania (parametr ‘nomatch’).

1.5.15 Operatory

Operatory dodawania			
Operacja +	data operator + (const data &a,const T t) data operator + (const T t,const data &a) data operator + (const data &a,const data &b)	d+1.5 “ABC”+d d+e	Dodawanie elementu 't' do obiektu 'a'. Element (liczba lub tekst) jest dodawany do każdego elementu obiektu. Dodawanie dwóch obiektów o zgodnych rozmiarach ('a' do 'b') . Dodawanie odbywa się po pozycjach. W przypadku liczb operacja + sumuje wartości. W przypadku tekstów operacja + scala teksty.
Operacja +=	void operator +=(const data &a) void operator +=(const T t)	d+=e d+=1.5 d+=”ABC”	Dodawanie analogiczne do +, ale przyrostowo na obiekcie.
Operatory porównania dwóch obiektów			

Porównanie	int compare<>(const data &a,const data &b)	compare(d,e)	Zwraca wynik porównania obiektów 'a' i 'b'. Jeśli jednakowe, zwraca 1 (true). Jeśli różne, zwraca 0 (false).
==, !=, <, >, <=, >=	vector operator == (const data &a,const data &b) itd.	d <= e d > e	Porównanie wektorów 'a' i 'b' po pozycjach. Zwraca wektor zero-jedynkowy (0 – false, 1 – true).
Operatory porównania obiektu ze stałą			
==, !=, <, >, <=, >=	vector operator ==(const data &a,const U u) itd.	d <= 1.5 d <= “ABC” d <= Text(“ABC”) d <= d(“2012-01-01”)	Porównanie wektora 'a' ze stałą 'u', która może być typu liczba, tekst lub data. Zwraca wektor zero-jedynkowy (0 – false, 1 – true).

1.5.16 Sortowanie

Sortowanie			
Sortowanie wektora	vector sort(data &v,Direction dir=ASCENDING,int sort_labels=1)	sort(d)	Sortowanie wektora ‘v’. Można wybrać kierunek sortowania (parametr ‘dir’) i czy sortować etykiety (parametr ‘sort_labels’). Możliwe kierunki sortowania ASCENDING (rosnąco) i DESCENDING (malejąco).
Sortowanie dualne	void sort(data &v1,data &v2,Direction dir=ASCENDING,int sort_labels=1)	sort(d,e)	Sortowanie dualne wektorów. Sortowany jest wektor ‘v1’. Kolejność przypadków w ‘v2’ jest taka sama jak kolejność posortowanego ‘v1’. Można wybrać kierunek sortowania (parametr ‘dir’) i czy sortować etykiety (parametr ‘sort_labels’)
Sortowanie danych	void sort(int no,Direction dir=ASCENDING,int	d.sort(1,DESCENDING,	Sortowanie całych danych. Można określić numer lub nazwę wiersz/kolumny (parametr ‘no’/’name’),

	sort_rows=1) void sort(Text name,Direction dir=ASCENDING,int sort_rows=1)	0) d.sort("WIEK",DESCENDING,1)	kierunek sortowania ('dir') oraz czy sortować wiersze, czy kolumny ('sort_rows'). W pierwszym przypadku 'd' jest sortowana malejąco. Na podstawie pierwszego wiersza są sortowane kolumny. W drugim przypadku 'd' jest sortowana malejąco. Na podstawie kolumny „WIEK" są sortowane wiersze.
Kolejność danych	vector order(const data &v,Direction dir=ASCENDING)	order(d)	Zwracany jest wektor kolejności po posortowaniu 'v'. Można wybrać kierunek sortowania (parametr 'dir').

1.5.17 Operacje na zbiorach

Operacja na zbiorach			
Elementy unikalne	data unique(const data &v,int fromlast=0)	unique(d)	Zwraca elementy unikalne wektora danych. Można wybrać kolejność działania funkcji – od początku wektora lub od końca (parametr 'fromlast')
Czy element	int iselement(T t,const data &v)	iselement("KOT",d)	Sprawdza, czy dany element 't' należy do zbioru 'v'. Zwraca 0 - false lub 1 – true
Czy elementy	vector iselement(const data &v1,const data &v2)	iselement(e,d)	Sprawdza, czy elemeny 'v1' należą do zbioru 'v2'. Zwraca wektor zero-jedynkowy (0 - false lub 1 – true)
Suma zbiorów	data setunion(const data &,const data &)	setunion(d,e)	Zwraca sumę zbiorów
Iloczyn zbiorów	data setintersect(const data &,const data &)	setintersect(d,e)	Zwraca iloczyn zbiorów
Różnica zbiorów	data setdiff(const data	setdiff(d,e)	Zwraca różnicę zbiorów

	&,const data &)		
Czy równe zbiory	int setequal(const data &,const data &)	setequal(d,e)	Zwraca 1 (true), jeśli zbiory są równe. Zwraca 0 (false), jeśli zbiory są różne

1.5.18 Operacje plikowe

Operacje plikowe			
Wczytanie danych z pliku tekstowego / zapis do pliku tekstowego	int readtxt(const char *name,int show_progress=0,int row_lab=1,int col_lab=1,int quote=0,char col_sep='\t',int show_open_err=1) int writetxt(const char *name,int show_progress=0,int row_lab=1,int col_lab=1,int quote=0,char col_sep='\t',char dec_sep=',',int show_open_err=1)	d.readtxt("plik.txt") d.writetxt("plik.txt")	Odczyt / zapis danych. Możliwe parametry: - name – nazwa pliku - show_progress – czy pokazywać postęp odczytu/zapisu (domyślnie nie) - row_lab – czy plik z etykietami wierszy (domyślnie tak) - col_lab – czy plik z etykietami kolumn (domyślnie tak) - quo te – czy elementy ujęte w cudzysłów (domyślnie nie) - col_sep – separator kolumn (domyślnie tabulator) - dec_sep – separator dziesiętny (domyślnie przecinek) - show_open_err – czy komunikować błędy (domyślnie tak) Funkcje zwracają 0, jeśli nie ma błędu. W innym przypadku 1.
Bezpieczne wczytanie danych z pliku tekstowego / bezpieczny zapis	void sreadtxt(const char *name,int show_progress=0,int row_lab=1,int	d.sreadtxt("plik.txt") d.swritetxt("plik.txt")	Bezpieczny odczyt / zapis. Jeśli plik jest zajęty, funkcja czeka na jego zwolnienie. Parametry jak przy zwykłym odczycie / zapisie. Inne parametry: - repeat_time – czas po jakim następują próby dostępu do pliku (domyślnie 1 sek.)

danych do pliku tekstowego	col_lab=1,int quote=0,char col_sep='\t',int repeat_time=1) void swritetxt(const char *name,int show_progress=0,int row_lab=1,int col_lab=1,int quote=0,char col_sep='\t',char dec_sep=',',int repeat_time=1)		

1.5.19 Wyświetlanie, kasowanie

Wyświetlanie, kasowanie			
Wyświetlanie	void display(Text name="")	d.display()	Obiekt jest wyświetlany w konsoli. Można podać jego nazwę (parametr ‘name’).
Kasowanie			
Kasowanie	void free()	d.free()	Obiekt jest kasowany. Zwalniana jest pamięć, z której korzystał.

1.6 Macierz (*matrix*)

1.6.1 Deklaracja zmiennej

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Utworzenie pustej macierzy	matrix()	matrix m	Tworzona jest pusta macierz
Utworzenie macierzy o zadanych wymiarach	matrix(int m,int n)	matrix m(5,3)	Tworzona jest macierz o wymiarach m x n (m wierszy, n kolumn). W tym przypadku tworzona jest macierz 5x3 (5 wierszy, 3 kolumny).
Utworzenie macierzy o zadanym wymiarze	matrix(int m)	matrix m(3)	Tworzona jest macierz wymiarach m x m. W tym przypadku tworzona jest macierz 3x3.
Utworzenie macierzy z liczby	matrix(double c)	matrix m(3.12)	Tworzona jest macierz o wymiarach 1x1, z elementem równym 'c'. W tym przypadku element macierzy 1x1 ma wartość 3.12.
Utworzenie macierzy na podstawie napisu	matrix(const char *matrix_str)	matrix m=”1 2; 3 4” m=”1 2; \ 3 4”	Tworzona jest macierz na podstawie napisu. Elementy w wierszu oddziela się spacjami, a średnik oznacza przejście do nowego wiersza. W pierwszym przykładzie powstaje macierz o wymiarach 2x2. Tę samą macierz dzięki symbolowi kontynuacji \ można zadeklarować w dwóch wierszach i wówczas jej zapis jest zgodny z naturalnym.

1.6.2 Konwersja

Konwersja			
Konwersja na liczbę całkowitą	operator int()	int(d)	Macierz o wymiarach 1x1 zamieniana jest na liczbę całkowitą. Jeśli ma inne wymiary, pojawia się komunikat błędu
Konwersja na liczbę rzeczywistą	operator double()	double(d)	Macierz o wymiarach 1x1 zamieniana jest na liczbę rzeczywistą. Jeśli ma inne wymiary, pojawia się komunikat błędu
Konwersja na wektor liczb całkowitych	vector as_vector(const matrix &m)	as_vector(d)	Jeśli macierz 'm' ma jeden z wymiarów równy 1 (jest wektorem), to jest zwracany wektor liczb całkowitych (*vector*) utworzony na jej podstawie. Jeśli 'm' ma inne wymiary, pojawia się komunikat błędu. Rzeczywiste wartości elementów są konwertowane na liczby całkowite. Obiekt wynikowy nie ma etykiet.

1.6.3 Tworzenie

Tworzenie wektorów			
Wektor stałych	matrix vd(intl ,...)	vd(4, 0.2, 1.5, -3.85, 10.0)	Tworzenie wektora wierszowego o długości l, wypełnionego podanymi stałymi. W tym przypadku powstaje wektor [0.2, 1.5, -3.85, 10.0] Jako stałe należy podawać liczby rzeczywiste (musi wystąpić kropka).
Sekwencje liczb	matrix seq(double a,double b) seq(double a,double step,double b) matrix seq(int a,int b) matrix seq(int a,int step,int b)	seq(1,9) seq(1,2,9)	Tworzenie wektora wierszowego zawierającego sekwencję liczb od 'a' do 'b', krokiem 1 lub 'step'. W pierwszym przypadku powstaje wektor [1, 2, 3, 4, 5, 6, 7, 8, 9] W drugim przypadku powstaje wektor [1, 3, 5, 7, 9]
Wektory liniowe	matrix linspace(double	linspace(1,2,5)	Tworzenie wektora o długości n, zawierającego liczby z przedziału od 'a' do 'b'. Domyślnie długość wynosi

	a,double b,int n=100)		100. W tym przypadku powstaje wektor [1.0, 1.25, 1.5, 1.75, 2.0]
Tworzenie macierzy			
Macierz zerowa	matrix zeros(int m,int n) matrix zeros(int m)	zeros(5,3) zeros(5)	Tworzenie macierzy o wymiarach m x n wypełnionej zerami. Tworzenie macierzy o wymiarach m x m wypełnionej zerami.
Macierz jedynkowa	matrix ones(int m,int n) matrix ones(int m)	ones(5,3) ones(5)	Tworzenie macierzy o wymiarach m x n wypełnionej jedynkami. Tworzenie macierzy o wymiarach m x m wypełnionej jedynkami.
Macierz z rozkładu jednostajnego	matrix rand(int m,int n) matrix rand(int m)	rand(5,3) rand(5)	Tworzenie macierzy o wymiarach m x n wypełnionej liczbami z rozkładu jednostajnego z przedziału [0,1]. Tworzenie macierzy o wymiarach m x m wypełnionej liczbami z rozkładu jednostajnego z przedziału [0,1].
Macierz z rozkładu normalnego	matrix randn(int m,int n) matrix randn(int m)	randn(5,3) randn(5)	Tworzenie macierzy o wymiarach m x n wypełnionej liczbami z rozkładu normalnego o parametrach (0,1). Tworzenie macierzy o wymiarach m x m wypełnionej liczbami z rozkładu normalnego o parametrach (0,1).
Macierz identycznościowa	matrix eye(int m,int n) matrix eye(int m)	eye(5,3) eye(5)	Tworzenie macierzy o wymiarach m x n wypełnionej zerami, a na głównej przekątnej jedynkami. Tworzenie macierzy o wymiarach m x m wypełnionej zerami, a na głównej przekątnej jedynkami.
Macierz diagonalna / Przekątna główna	matrix diag(const matrix &a)	m = diag(v) v = diag(m)	1) Jeśli 'a' jest wektorem, to tworzona jest macierz kwadratowa wypełniona zerami, a na głównej przekątnej wektorem 'a'. 2) Jeśli 'a' jest macierzą, to tworzeny jest wektor na podstawie przekątnej macierzy 'a'.
Tworzenie na podstawie wektorów i funkcji arytmetycznej	matrix outer(const matrix &v1,const matrix &v2,char fun)	outer(c(1,2,3,4),c(1,2,3,4),'*')	Tworzona jest macierz, której elementy są wynikiem funkcji zdefiniowanej znakiem 'fun' z argumentami pochodzącymi z wektora v1 i v2. Możliwe są funkcje: +, -, *, /. Elementy obiektu wyjściowego o współrzędnych (i,j) przyjmują wartości fun(v1(i),v2(j)). W tym przypadku zwracana jest tabliczka mnożenia do 16.

1.6.4 Wypełnianie

Wypełnianie			
Zerowanie	void zeros()	m.zeros()	Macierz jest zerowana.
Wypełnianie jedynkami	void ones()	m.ones()	Macierz jest wypełniana jedynkami.
Wypełnianie wartościami z rozkładu jednostajnego	void rand()	m.rand()	Macierz jest wypełniana wartościami z rozkładu jednostajnego [0,1].
Wypełnianie wartościami z rozkładu normalnego	void randn()	m.randn()	Macierz jest wypełniana wartościami z rozkładu normalnego o parametrach (0,1).
Wypełnianie identycznościowe	void eye()	m.eye()	Macierz jest wypełniana zerami, a na głównej przekątnej jedynkami.
Wypełnianie wartościami testowymi	void test(double t=0.0,Text e="")	m.test()	Macierz jest wypełniana wartościami testowymi (kolejne liczby całkowite). Etykiety przyjmują standardowe nazwy (R1, R2, itd. C1, C2 itd.). Wartości mogą być przesunięte o stałą 't', a etykiety mogą mieć dołączony z przodu człon 'e'.

1.6.5 Podstawowe operacje

Podstawowe operacje			
Wartość bezwzględna	matrix abs(const matrix &a) void abs()	abs(m) m.abs()	Tworzona jest macierz wypełniona wartościami bezwzględnymi elementów 'a'. Nakładana jest funkcja abs na elementy.

Podniesienie elementów do kwadratu	matrix sqr(const matrix &a) void sqr()	sqr(m) m.sqr()	Tworzona jest macierz wypełniona kwadratami elementów 'a'. Nakładana jest funkcja ^2 na elementy.
Unormowanie	matrix norm(const matrix &a) void norm()	norm(m) m.norm()	Tworzona jest macierz z unormowanymi elementami 'a'. Unormowane wartości są z przedziału [0,1]. Macierz jest normowana.
Minimum	matrix min(const matrix &a)	min(m)	Znajdowane jest minimum elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Maksimum	matrix max(const matrix &a)	max(m)	Znajdowane jest maksimum elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Licznik	matrix count(const matrix &a)	count(m)	Zliczane są elementy liczbowe różne od NAN każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Suma	matrix sum(const matrix &a)	sum(m)	Wyliczana jest suma elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Suma kwadratów	matrix sumsqr(const matrix &a)	sumsqr(m)	Wyliczana jest suma kwadratów elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Iloczyn	matrix prod(const matrix &a)	prod(m)	Wyliczany jest iloczyn elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Suma skumulowana	matrix cumsum(const matrix &a)	cumsum(m)	Wyliczana jest suma skumulowana elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym).
Iloczyn skumulowany	matrix cumprod(const matrix &a)	cumprod(m)	Wyliczany jest iloczyn skumulowany elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym).
Przeciwieństwo	void minus()	m.minus()	Elementy macierzy są zamieniane na przeciwne.

1.6.6 Operacje statystyczne

Operacje statystyczne			
Wartość średnia (oczekiwana)	matrix mean(const matrix &a)	mean(m)	Obliczana jest wartość średnia elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Wariancja	matrix var(const matrix &a)	var(m)	Obliczana jest wariancja elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Odchylenie standardowe	matrix stdev(const matrix &a)	stdev(m)	Obliczane jest odchylenia standardowe elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Mediana	matrix median(const matrix &a)	median(m)	Obliczana jest mediana elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Moda	matrix mode(const matrix &a)	mode(m)	Obliczana jest moda elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1).
Kwantyl / kwantyle	matrix quantile(const matrix &a,double c) matrix quantile(const matrix &a,const matrix &v)	quantile(m,0.5) quantile(m,c(0.25, 0.75))	Obliczany jest kwantyl elementów każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym). W wyniku zwracany jest wektor (lub macierz 1x1). Można wyliczyć kwantyl dla jednej wartości 'c' lub kilka kwantyli zdefiniowanych w wektorze 'v'. W pierwszym przypadku obliczany jest kwantyl 50%. W drugim przypadku obliczany są kwantyle 25% i 75%.
Crossproduct	matrix crossprod(const matrix &a)	crossprod(m)	Obliczana jest suma iloczynów każdej kolumny z każdą macierzy 'a'.
Kowariancja	matrix cov(const matrix &a)	cov(m)	Obliczana jest kowariancja każdej kolumny z każdą macierzy 'a'.

Korelacja	matrix cor(const matrix &a)	cor(m)	Obliczana jest korelacja każdej kolumny z każdą macierzy 'a'.
Histogram	matrix hist(const matrix &a)	hist(m)	Wyznaczany jest histogram dla każdej kolumny. Wynik jest macierzą z podanymi udziałami w przedziałach.

1.6.7 Ogólne podsumowanie

Operacje statystyczne			
Podsumowanie kolumn	matrix summarise(double f(const matrix &))	m.summarise(fun1)	Dla każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym) wyliczania jest wartość funkcji f. Funkcja f ma działać na wektorze i zwracać jedną liczbę. W wyniku zwracany jest wektor (lub macierz 1x1).
Podsumowanie kolumn z parametrem	matrix summarise(double f(const matrix &,double),double)	m.summarise(fun2, 3.5)	Dla każdej kolumny (lub wiersza, jeśli macierz jest wektorem wierszowym) wyliczania jest wartość funkcji f. Funkcja f ma działać na wektorze, posiadać dodatkowy parametr liczbowy i zwracać jedną liczbę. W wyniku zwracany jest wektor (lub macierz 1x1).

1.6.8 Operacje dwuargumentowe

Operacje dwuargumentowe			
Minimum dwuargumentowe	matrix min(const matrix &a,const matrix &b) matrix min(const matrix &a,const double c) matrix min(const double c,const matrix &a)	min(m1,m2) min(m,20)	Minimum po pozycjach macierzy 'a' i 'b'. Minimum macierzy 'a' i stałej 'c'.

Maksimum dwuargumentowe	matrix max(const matrix &a,const matrix &b) matrix max(const matrix &a,const double c) max(const double c,const matrix &a)	max(m1,m2) max(m,20)	Maksimum po pozycjach macierzy 'a' i 'b'. Maksimum macierzy 'a' i stałej 'c'.
Kowariancja dwuargumentowa	double cov(const matrix &a,const matrix &b)	cov(m1,m2)	Kowariancja wektorów 'a' i 'b'. W wyniku zwracana jest liczba.
Korelacja dwuargumentowa	double cor(const matrix &a,const matrix &b)	cor(m1,m2)	Korelacja wektorów 'a' i 'b'. W wyniku zwracana jest liczba.
Suma warunkowa	matrix sumif(const matrix &a,const vector &v)	sumif(m,v)	Suma warunkowa elementów w kolumnach macierzy 'a'. Wektor 'v' definiuje, które wiersze brać pod uwagę przy sumowaniu. Zwracany jest wektor wierszowy.
Średnia warunkowa	matrix meanif(const matrix &a,const vector &v)	meanif(m,v)	Średnia warunkowa elementów w kolumnach macierzy 'a'. Wektor 'v' definiuje, które wiersze brać pod uwagę przy uśrednianiu. Zwracany jest wektor wierszowy.

1.6.9 Operatory

Operatory i funkcje arytmetyczne			
+, -, *, /	matrix operator +(const matrix &a,const matrix &b) matrix operator +(const matrix &a,const double c)	m1+m2 m+10.0 m1+=m2 m+=10.0	Umożliwiają wykonywanie operacji arytmetycznych na dwóch macierzach lub macierzy i liczbie rzeczywistej. Operacje +,-,*,/ są wykonywane po pozycjach.

	matrix operator +(const double c,const matrix &a) void operator +=(const matrix &a) void operator +=(const double c)		
Mnożenie macierzowe	matrix mul(const matrix &a,const matrix &b) matrix mulT(const matrix &a,const matrix &b)	mul(m1,m2) mulT(m1,m2)	Mnożenie macierzowe. Mnożenie macierzowe z transpozycją. Odpowiada operacji mul(a,~b)
Podniesienie do kwadratu z transpozycją	matrix sqrT(const matrix &a)	sqrT(m)	Podniesienie do kwadratu z transpozycją. Odpowiada operacji mul(a,~a).
Podniesienie do kwadratu z transpozycją 2	matrix sqrT2(const matrix &a)	sqrT2(m)	Podniesienie do kwadratu z transpozycją wersja 2. Odpowiada operacji mul(~a,a).
Iloczyn skalarny	double scmul(const matrix &a,const matrix &b)	scmul(m1,m2)	Iloczyn skalarny.
Kwadrat skalarny	double scsqr(const matrix &a)	scsqr(m)	Kwadrat skalarny.
Komutator	double comut(const matrix &a,const matrix &b)	comut(m1,m2)	Komutator.
Ślad	double trace(const matrix &a)	trace(m)	Ślad macierzy.

Funkcje matematyczne			
Sinus	matrix sin(const matrix &a) void sin()	sin(m) m.sin()	Sinus po elementach macierzy.
Cosinus	matrix cos(const matrix &a) void cos()	cos(m) m.cos()	Cosinus po elementach macierzy.
Tangens	matrix tan(const matrix &a) void tan()	tan(m) m.tan()	Tangens po elementach macierzy.
Sinus hiperboliczny	matrix sinh(const matrix &a) void sinh()	sinh(m) m.sinh()	Sinus hiperboliczny po elementach macierzy.
Cosinus hiperboliczny	matrix cosh(const matrix &a) void cosh()	cosh(m) m.cosh()	Cosinus hiperboliczny po elementach macierzy.
Tangens hiperboliczny	matrix tanh(const matrix &a) void tanh()	tanh(m) m.tanh()	Tangens hiperboliczny po elementach macierzy.
Arcus sinus	matrix asin(const matrix &a) void asin()	asin(m) m.asin()	Arcus sinus po elementach macierzy.

Arcus cosinus	matrix acos(const matrix &a) void acos()	acos(m) m.acos()	Arcus cosinus po elementach macierzy.
Arcus tangens	matrix atan(const matrix &a) void atan()	atan(m) m.atan()	Arcus tangens po elementach macierzy.
Eksponenta	matrix exp(const matrix &a) void exp()	exp(m) m.exp()	Eksponenta po elementach macierzy.
Logarytm	matrix log(const matrix &a) void log()	log(m) m.log()	Logarytm po elementach macierzy.
Logarytm dziesiętny	matrix log10(const matrix &a) void log10()	log10(m) m.log10()	Logarytm dziesiętny po elementach macierzy.
Potęga	matrix pow(const matrix &a, double d) void pow(double d)	pow(m,3.0) m.pow(3.0)	Potęga po elementach macierzy.
Potęga całkowita	matrix pow(const matrix &a, int d) void pow(int d)	pow(m,3) m.pow(3)	Potęga całkowita po elementach macierzy.
Pierwiastek	matrix sqrt(const matrix	sqrt(m)	Pierwiastek po elementach macierzy.

	&a) void sqrt()	m.sqrt()	
Zaokrąglenie w górę	matrix ceil(const matrix &a) void ceil()	ceil(m) m.ceil()	Zaokrąglenie w górę po elementach macierzy.
Zaokrąglenie w dół	matrix floor(const matrix &a) void floor()	floor(m) m.floor()	Zaokrąglenie w dół po elementach macierzy.
Operatory logiczne			
Zaprzeczenie	matrix operator !(const matrix &a)	!m	Zaprzeczenie elementów macierzy. Zera zamieniane są na jedynki, inne wartości na zera.
Koniunkcja	matrix operator &&(const matrix &a,const matrix &b)	m1&&m2	Koniunkcja po elementach macierzy 'a' i 'b'
Alternatywa	matrix operator ||(const matrix &a,const matrix &b)	m1||m2	Alternatywa po elementach macierzy 'a' i 'b'
Operatory porównania			
==, !=, <, >, <=, >=	matrix operator ==(const matrix &a,const matrix &b) itd.	m1 == m2 m1 < m2	Operatory porównania umożliwiają porównanie macierzy 'a' i 'b' po pozycjach. Macierze nie muszą być wektorami, jak to jest w przypadku operatorów dziedziczonych z obiektu *data*.

1.6.10 Algebra liniowa

Projekcja i norma			
Projekcja	matrix proj(const matrix &a,const matrix &b)	proj(m1,m2)	Projekcja 'a' na 'b'
Norma	double norm2(const matrix &a)	norm2(m)	Norma w przestrzeni R2.
Ortogonalizacja i dekompozycja			
Ortonormalizacja	void orthonorm()	m.orthonorm()	Ortogonalizacja Grama-Schmidta z normalizacją
Dekompozycja QR	void QR_decomp(const matrix &a,matrix &Q,matrix &R)	QR_decomp(m,Q,R)	Dekompozycja macierzy 'a' na macierz ortogonalną Q i trójkątną R
Dekompozycja LU	void LU_decomp(const matrix &a,matrix &LU,matrix &pivot,int &pivsign)	LU_decomp(m,LU,pivot,pivsign)	Dekompozycja macierzy 'a' na dwie macierze trójkątne L i U (dolną i górną). W wyniku zwracane jest złożenie LU, macierz przestawień 'pivot' i znak przestawień 'pivsign'.
Układy równań			
Wyznacznik	double det(const matrix &a)	det(m)	Wyznacznik macierzy.
Rozwiązywanie układu równań	matrix lineqns(const matrix &a,const matrix &b,LinearMethod met=GAUSS)	lineqns(m,v)	Zwraca rozwiązanie układu równań liniowych danego macierzą 'a' i kolumną 'b'. Za pomocą parametru 'met' można wybrać metodę {GAUSS, QR_DECOMP, LU_DECOMP}. Domyślnie jest stosowana metoda eliminacji Gaussa.
Macierz odwrotna	matrix inv(const matrix &a,LinearMethod	inv(m)	Macierz odwrotna. Za pomocą parametru 'met' można wybrać metodę {GAUSS, QR_DECOMP, LU_DECOMP}. Domyślnie jest stosowana dekompozycja LU.

	met=LU_DECOMP)		

1.6.11 Regresja liniowa i inne metody numeryczne

Regresja liniowa			
Sweeping	void linreg_sweep(int k1,int k2)	m.linreg_sweep(1,10)	Sweeping macierzy dla kolumn od k1 do k2.
Regresja liniowa	matrix linreg(const matrix &a,const matrix &b)	linreg(m,v)	Regresja liniowa dla zmiennych objaśniających 'a' i zmiennej objaśnianej 'b'.
Sweeping warunkowy	void linreg_sweep_c(int k1,int k2,const matrix &cond,int dep)	m.linreg_sweep_c(1,10,v,10)	Sweeping macierzy dla kolumn od k1 do k2, wskazanych przez wektor zero-jedynkowy 'cond'. Parametr 'dep' określa numer zmienne objaśnianej.
Regresja liniowa warunkowa	matrix linreg_c(const matrix &cross,const matrix &cond,int dep)	linreg_c(iloczyny,wybrane,10)	Regresja liniowa. Parametr cross to macierz iloczynów zmiennych objaśniających i zmiennej objaśnianej. Parametr cond to wektor zero-jedynkowy wskazujący zmienne do wykorzystania, 'dep' określa numer zmiennej objaśniającej w macierzy cross.
Wygładzanie			
Wygładzanie	matrix smooth(const matrix &v,int m,int n);	smooth(v,1,3)	Wygładzanie za pomocą wielomianów interpolacyjnych. Parametr 'm' określa stopień wielomianu interpolacyjnego (od 1 do 5), parametr 'n' liczbę punktów do uśrednienia (oczekiwana jest liczba nieparzysta). W tym przypadku wyznaczana jest 3-punktowa średnia ruchoma.

1.6.12 Operacje na plikach binarnych

Regresja liniowa

Zapis do pliku	void writebin(FILE *f)	m.writebin(f)	Macierz jest zapisywana do pliku.
Odczyt z pliku	void readbin(FILE *f)	m.readbin(f)	Macierz jest odczytywana z pliku.

1.7 Tabela (*table*)

1.7.1 Deklaracja zmiennej

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Utworzenie pustej tabeli	table()	table t	Tworzona jest pusta tabela
Utworzenie tabeli o zadanych wymiarach	table(int m,int n)	table t(5,3)	Tworzona jest tabela o wymiarach m x n (m wierszy, n kolumn). W tym przypadku tworzona jest tabela 5x3 (5 wierszy, 3 kolumny).

1.7.2 Konwersja

Konwersja			
Konwersja na macierz	operator matrix()	matrix(t)	Tabela jest zamieniana w macierz. Aby operacja była możliwa, konieczne jest wypełnienie tablicy elementami tekstowymi, które konwertują się na liczby rzeczywiste.

1.7.3 Wybór etykiet

Wybór etykiet			
Wybór nazw kolumn	table colnames(const table &)	colnames(t)	Etykiety kolumn są zwracane jako tabela.
Wybór nazw wierszy	table rownames(const	rownames(t)	Etykiety wierszy są zwracane jako tabela.

	table &)		

1.7.4 Tabela testowa

Tabela testowa			
Wypełnianie wartościami testowymi	void test(Text p="",Text e="")	t.test()	Tabela jest wypełniana wartościami testowymi (kolejne liczby całkowite). Etykiety przyjmują standardowe nazwy (R1, R2, itd. C1, C2 itd.). Elementy mogą być poprzedzone tekstem 'p', a etykiety mogą mieć dołączony z przodu człon 'e'.

1.7.5 Operacje wyboru i faktorowe

Operacje wyboru i faktorowe			
Wybór lub kasowanie kolumn lub wierszy	table select(const table &names,int cols=1,int del=0)	t.select(c(”C1”,”C3”))	Z tabeli są wybierane lub kasowanie wskazane kolumny lub wiersze. Jeżeli cols=1, to operacja dotyczy kolumn. Jeżeli cols=0, to operacja dotyczy wierszy. Jeżeli del=0, to następuje wybieranie. Jeżeli del=1, to następuje kasowanie.
Wybieranie kolumn o zadanych właściwościach	table select_if(int f(const table &))	t.select_if(funkcja)	Z tabeli są wybierane kolumny o zadanych właściwościach. Argument ‘f’ to dowolna funkcja działająca na pojedynczej kolumnie i zwracająca wartość logiczną 0 lub 1 (0 – odrzuć kolumnę, 1 – zostaw kolumnę). Przykładowo ‘f’ może wykrywać specyficzne wartości w kolumnie lub określać jej typ.
Podsumowanie w grupach	table summarise(const table &names,Text formula,matrix f(const matrix &)=sum, ...)	t.summarise(c(”ROK”, ”MARKA”),”SR_POJ=f(POJ)”,mean)	Funkcja grupuje dane według pól, których nazwy są zdefiniowane jako names. W grupach wyliczane jest podsumowanie według wzoru formula. Formułę definiuje się w postaci: „nazwa_wyniku = f(nazwa_zmiennej)”. Nazwa_wyniku to dowolna nazwa, która pojawi się jako kolumna w wyniku zwracanym przez summarise. Nazwa_zmiennej to nazwa dowolnej kolumny w tabeli, dla której wywołuje się podsumowanie. Funkcję podsumowującą definiuje parametr ‘f’ – może to być jedna ze standardowych funkcji macierzowych (count, sum, mean, min, max itd.) lub dowolna funkcja użytkownika. Domyślnie użyta będzie funkcja sum.

			W jednym wywołaniu summarise można podać do 10 formuł i funkcji i wyliczyć kilka podsumowań. W przykładzie wykonanie zostanie grupowanie tabeli ‘t’ według zmiennych ROK i MARKA i w grupach wyliczona zostanie zmienna SR_POJ jako średnia ze zmiennej POJ.
Przedziałowanie zmiennej	table cut(const table &X,const matrix &breaks,int right=1)	cut(t,c(1,2,3))	Zmienna dana wektorem X jest przedziałowana. Przedziały są zdefiniowane granicami przez parametr 'breaks'. Parametr 'right' określa, czy przedziały mają być prawostronnie domknięte (domyślnie tak). Wektor X może też być typu *matrix*.
Poziomy zmiennej	table levels(const table &X)	levels(t)	Wyznaczane są poziomy wektora X. Wektor X może też być typu *matrix*.
Tabela wystąpień	table pivot(const table &X) table pivot(const table &X1,const table &X2) table pivot(const table &X1,const table &X2,const table &X3)	pivot(t) pivot(t1,t2)	Wyliczana jest liczba wystąpień każdego poziomu wektora X. Jeśli argumentów jest więcej niż 1, to wyliczane są wystąpienia dla każdej kombinacji poziomów wszystkich argumentów. Wektory X mogą też być typu *matrix*.
Zastosowanie funkcji do danych	table apply(const table &X,const table &index,matrix f(const matrix &)) table apply(const table &X,const table &index,matrix f(const matrix &,double),double p)	apply(t,v,mean) apply(t,v,quantile,0.25)	W grupach zdefiniowanych przez 'index' wyliczane jest wartość funkcji f(X). Wektor X może też być typu *matrix*. W pierwszym przypadku wyliczane są średnie wartości 't' w grupach danych przez 'v'. W drugim przypadku wyliczane są pierwsze kwartyle wartości 't' w grupach danych przez 'v'. Funkcje mean i quantile są wbudowane w typ *matrix*. Można również wykorzystywać własne funkcje.

1.7.6 Sortowanie

Sortowanie			
Sortowanie według wartości	void sortval(int no,Direction dir=ASCENDING,int sort_rows=1) void sortval(Text name,Direction dir=ASCENDING,int sort_rows=1)	sortval(10) sortval(„WIEK”)	Tabela jest sortowana według pola o numerze 'no'. Tabela jest sortowana według pola o etykiecie name. W odróżnieniu od funkcji 'sort' dziedziczonej z typu *data*, to sortowanie jest przeprowadzane według wartości liczbowych, a nie według znakowych (liter). Można wybrać kierunek sortowania (parametr dir) oraz czy sortować wiersze, czy kolumny (parametr sort_rows). Możliwe kierunki sortowania: ASCENDING - rosnąco, DESCENDING – malejąco. Domyślnie rosnąco sortowane są wiersze.

1.7.7 Tworzenie bazy do modelowania

Tworzenie bazy do modelowania			
Tworzenie bazy numerycznej	table build_numerical_base(const table &input,BaseVer ver=NORM,int show_progress=0,Text na="0")	build_numerical_base(t)	Tworzona jest baza numeryczna do modelowania na podstawie tabeli 'input'. Można wybrać sposób przygotowania pól (parametr ver), czy pokazywać postęp (parametr show_progress) oraz jakie wartości użyć w miejsce braków danych (parametr na). Możliwe sposoby przygotowania to SIMP (prosty), NORM (normalny), EXT (rozszerzony). Domyślnie tworzona jest baza sposobem normalnym, bez pokazywania postępu. Dla sposobu SIMP w wyniku zapisywane są zmienne liczbowe uzupełnione zerami w miejscu braków i zmienne binarne utworzone ze zmiennych kategorialnych. Dla sposobu NORM w wyniku zapisywane są te same zmienne co dla sposobu SIMP oraz dodatkowo zmienne określające wypełnienie pól (dotyczy pól z brakującymi danymi). Dla sposoby EXT wszystkie zmienne są zamieniane na binarne. Zmienne kategorialne analogicznie jak w sposobie SIMP/NORM. Zmienne liczbowe są przedziałowane.

Tworzenie bazy konstrukcyjnej i walidacyjnej	void constr_and_valid(Text name_constr,Text name_valid,double percent_size,int show_progress=0)	t.constr_and_valid(„dane_konstr.txt”,”dane_walid.txt”,0.75)	Tabela 't' dzielona jest losowo na część konstrukcyjną i walidacyjną. Wyniki zapisywane są do plików o nazwach name_constr i name_valid. Należy określić udział danych konstrukcyjnych (parametr percent_size). Można określić, czy pokazywać postęp (parametr show_progress). Domyślnie postęp nie jest pokazywany.
Tworzenie bazy numerycznej o zadanych polach	void build_numerical_with_columns(table &tab_colnames,Text name_data,Text name_errors,int show_progress=0,Text na="0")	t. build_numerical_with_columns(nazwy_kolumn,”nowe_dane.txt”,”bledy.txt”)	Na podstawie tabeli 't' tworzona jest baza numeryczna o zadanych polach (parametr tab_colnames). Wynik zapisywany jest do pliku o nazwie name_data. Jeżeli w tabeli 't' brakuje pól niezbędnych do ukończenia operacji, pojawiają się błędy, które są zapisywane do pliku o nazwie name_errors. Można określić, czy pokazywać postęp (parametr show_progress). Domyślnie postęp nie jest pokazywany. Parametr 'na' wskazuje, jakie wartości zastosować w miejsce braków danych.

1.8 Macierz rzadka (*sparse*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Utworzenie pustej macierzy rzadkiej	sparse()	sparse s	Tworzona jest pusta macierz rzadka
Utworzenie macierzy rzadkiej o zadanych rozmiarach	sparse(int m,int n)	sparse s(10,20)	Tworzona jest macierz rzadka o wymiarach m x n (m wierszy, n kolumn)
Utworzenie macierzy rzadkiej na podstawie macierzy gęstej	sparse(const matrix &mat)	matrix m=zeros(5,5) sparse s=sparse(m)	Tworzona jest macierz rzadka na podstawie macierzy gęstej. W przykładzie tworzona jest macierz gęsta 'm', a następnie macierz rzadka 's'. Zarówno 'm' jak i 's' to macierze o wymiarach 5 x 5, wypełnione zerami.
Konwersja			
Konwersja macierzy rzadkiej na macierz gęstą	operator matrix()	sparse s=spzeros(5,5) matrix m=matrix(s)	Tworzona jest macierz gęsta na podstawie macierzy rzadkiej. W przykładzie tworzona jest macierz rzadka 's', a następnie macierz gęsta 'm'. Zarówno 's' jak i 'm' to macierze o wymiarach 5 x 5, wypełnione zerami.
Konwersja macierzy rzadkiej na liczbę	operator double()	double(s)	Macierz rzadka o wymiarach 1x1 zamieniana jest na liczbę rzeczywistą. Jeśli ma inne wymiary, pojawia się komunikat błędu
Informacja o rozmiarach			
Liczba wierszy	int size1(const sparse &a) int nrow(const sparse &a)	size1(s) nrow(s)	Zwraca liczbę wierszy macierzy
Liczba kolumn	int size2(const sparse &a)	size2(s)	Zwraca liczbę kolumn macierzy

	int ncol(const sparse &a)	ncol(s)	
Liczba elementów niezerowych (istotnych)	int nonzero(const sparse &a)	nonzero(s)	Zwraca liczbę niezerowych elementów macierzy
Czy zaindeksowana	int indexed(const sparse &a)	indexed(s)	Zwraca 1 (true), jeśli macierz ma indeks elementów; lub 0 (false), jeśli nie ma.
Zakresy niezerowe			
Najmniejszy wiersz istotny	int minrow(const sparse &a)	minrow(s)	Zwraca najmniejszy numer wiersza, w którym są elementy niezerowe
Największy wiersz istotny	int maxrow(const sparse &a)	maxrow(s)	Zwraca największy numer wiersza, w którym są elementy niezerowe
Najmniejsza kolumna istotna	int mincol(const sparse &a)	mincol(s)	Zwraca najmniejszy numer kolumny, w którym są elementy niezerowe
Największa kolumna istotna	int maxcol(const sparse &a)	maxcol(s)	Zwraca największy numer kolumny, w którym są elementy niezerowe
Informacja o wypełnieniu			
Czy element istnieje	int iselem(int i,int j)	iselem(2,7)	Zwraca 1 (true), jeśli element (i,j) występuje w macierzy; 0 (false), jeśli nie występuje. Elementom macierzy rzadkiej, które nie występują, odpowiada wartość 0.
Czy element jest niezerowy	int iszero(int,int)	iszero(2,7)	Zwraca 1 (true), jeśli element (i,j) macierzy ma wartość 0; 0 (false), w przeciwnym przypadku. True jest zwracane zarówno, gdy element nie występuje, jak i gdy występuje i przyjmuje wartość 0.
Indeksowanie			
Indeksowanie	void mkindex()	s.mkindex()	Macierz jest indeksowana. Indeks ułatwia wyszukiwanie elementów istotnych, dzięki czemu operacje na macierzy są wykonywane szybciej.

Kasowanie indeksu	void delindex()	s.delindex()	Indeks macierzy jest usuwany
Kopiowanie indeksu (niskopoziomowe)	void cpyindex(const sparse &src)	s.cpyindex(s2)	Macierz otrzymuje indeks skopiowany z macierzy src. Wymagane jest, aby obie macierze (źródłowa i docelowa) miały te same wymiary; oprócz tego trzeba mieć pewność, że pozycje niezerowe w obu macierzach są na tych samych pozycjach.
Zmiana rozmiaru			
Zmiana rozmiaru	void resize(int m_new,int n_new)	s.resize(5,10)	Zmiana wymiarów macierzy. Po zmianie macierz ma m_new wierszy i n_new kolumn. Zawartość macierzy ulega skasowaniu.
Zmiana rozmiaru z zachowaniem zawartości	void resizec(int m_new,int n_new)	s.resizec(5,10)	Zmiana wymiarów macierzy. Po zmianie macierz ma m_new wierszy i n_new kolumn. Zawartość macierzy jest zachowywana. Jeśli nowe wymiary są większe od pierwotnych, to elementy nie występujące w obiekcie pierwotnym pozostają niewypełnione.
Pakowanie i synchronizacja elementów			
Pakowanie	void pack()	s.pack()	Macierz jest pakowana. Po operacji pozostają tylko elementy istotne (niezerowe).
Odpakowanie	void unpack()	s.unpack()	Macierz jest rozpakowywana. Po operacji wszystkie elementy nieistotne są reprezentowane w obiekcie zerami.
Synchronizacja	void synch(const sparse &src)	s.synch(s2)	Macierz jest synchronizowana z macierzą src. Uzgadniana jest kolejność elementów istotnych na liście, dzięki czemu operacje, których argumentami jest macierz źródłowa i uzgodniona, są wykonywane szybciej (patrz → arytmetyka macierzy zsynchronizowanych).
Dostęp do elementu			
Dostęp do elementu	double & operator ()(int i,int j)	s(2,3)	Zwraca element z wiersza 'i' i kolumny 'j'.
Szybki dostęp do elementu	double get(int i,int j)	s.get(2,3)	Zwraca element z wiersza 'i' i kolumny 'j'. Funkcja działa szybciej niż operator ().

Szybkie wstawienie wartości	void set(int i,int j,double d)	s.set(2,3,11.5)	Wstawia wartość 'd' do elementu z wiersza 'i' i kolumny 'j'. Funkcja działa szybciej niż operator ().
Dostęp do listy elementów (niskopoziomowo)	double & operator [](int l)	s[15]	Zwraca element o numerze 'l' z listy elementów istotnych macierzy.
Tworzenie			
Macierz pusta	sparse spempty(int m,int n)	spempty(5,3)	Tworzenie pustej macierzy rzadkiej o wymiarach m x n.
Macierz zerowa	sparse spzeros(int m,int n)	spzeros(5,3)	Tworzenie macierzy rzadkiej o wymiarach m x n wypełnionej zerami.
Macierz jedynkowa	sparse spones(int m,int n)	spones(5,3)	Tworzenie macierzy rzadkiej o wymiarach m x n wypełnionej jedynkami.
Macierz identycznościowa	sparse speye(int m,int n)	speye(5,3)	Tworzenie macierzy rzadkiej o wymiarach m x n wypełnionej zerami, a na głównej przekątnej jedynkami.
Macierz testowa	sparse sptest(int m,int n)	sptest(5,3)	Tworzenie macierzy rzadkiej o wymiarach m x n wypełnionej wartościami testowymi (kolejne liczby naturalne)
Macierz losowa	sparse sprand(int m,int n)	sprand(5,3)	Tworzenie macierzy rzadkiej o wymiarach m x n wypełnionej liczbami z rozkładu jednostajnego z przedziału [0,1].
Wypełnianie			
Wypełnianie zerami	void zeros()	s.zeros()	Istotne elementy są zerowane
Wypełnianie jedynkami	void ones()	s.ones()	Istotne elementy przyjmują wartość 1
Wypełnianie	void fill(double c=1.0)	s.fill(2.0)	Istotne elementy są wypełnianie stałą c
Wypełnianie zer	void fill0(double c=1.0)	s.fill0(2.0)	Istotne elementy wynoszące 0 są wypełnianie stałą c
Losowe wypełnianie	void rfill(double c=1.0)	s.rfill(3.12)	Istotne elementy są wypełnianie liczbą losową z rozkładu jednostajnego [0,c]

Losowe wypełnianie zer	void rfill0(double c=1.0)	srfill0(1.5)	Istotne elementy wynoszące 0 są wypełnianie liczbą losową z rozkładu jednostajnego [0,c]
Aplikowanie funkcji			
Aplikowanie funkcji do elementów	sparse forall(const sparse &s,double f(double)) void forall(double f(double))	forall(s,f1) s.forall(f1)	Na elementy macierzy nakładana jest funkcja 'f'
Aplikowanie funkcji dwuargumentowej do elementów	sparse forall(const sparse &s,double f(double,double),double d) void forall(double f(double,double),double d) sparse forall(const sparse &s,double f(double,int),int i) void forall(double f(double,int),int i)	forall(s,f1,3.52) s.forall(f1,3.52) forall(s,f1,82) s.forall(f1,82)	Na elementy macierzy nakładana jest funkcja dwuargumentowa 'f'. Drugi argument funkcji 'f' (parametr) jest przekazywany do funkcji forall.
Podstawowe operacje			
Wartość bezwzględna	sparse abs(const sparse &a) void spabs()	abs(s) s.spabs()	Tworzona jest macierz rzadka wypełniona wartościami bezwzględnymi elementów 'a'.
Podniesienie elementów do kwadratu	sparse sqr(const sparse &a) void spsqr()	sqr(s) s.spsqr()	Tworzona jest macierz rzadka wypełniona kwadratami elementów 'a'.

Suma	double sum(const sparse &a)	sum(s)	Wyliczana jest suma wszystkich elementów macierzy 'a'.
Suma kwadratów	double sumsqr(const sparse &a)	sumsqr(s)	Wyliczana jest suma kwadratów wszystkich elementów macierzy 'a'.
Operacje arytmetyczne			
+, -	sparse operator +(const sparse &a,const sparse &b) void operator +=(const sparse &a) void operator +=(const double d)	s1+s2 s1+=s2 s1+=10.0	Umożliwiają wykonywanie operacji arytmetycznych na dwóch macierzach rzadkich lub macierzy rzadkiej i liczbie rzeczywistej. Operacje +, - są wykonywane po pozycjach.
***, /**	sparse operator *(const sparse &a,const double d) sparse operator *(const double d,const sparse &a) void operator *=(const double d) sparse operator *(const sparse &a,const sparse &b)	s1*10.0 10.0*s1 s1*=10.0 s1*s2	Umożliwiają wykonywanie operacji arytmetycznych na dwóch macierzach rzadkich lub macierzy rzadkiej i liczbie rzeczywistej. Operacje *, / są wykonywane po pozycjach.
Mnożenie macierzowe	sparse mul(const sparse &a,const sparse &b)	mul(s1,s2)	Mnożenie macierzowe
Arytmetyka macierzy zsynchronizowanych			
Dodawanie	sparse spplus(const sparse	spplus(s1,s2)	Dodawanie macierzy: a+b

	&a,const sparse &b)		
Zwiększanie	void addassign(const sparse &a)	s.addassign(s2)	Zwiększanie macierzy: baza=baza+a
Zwiększanie i mnożenie (wersja 1)	void addassign(double d,const sparse &a)	s.addassign(10.0,s2)	Zwiększanie i mnożenie macierzy: baza=baza*d+a
Zwiększanie i mnożenie (wersja 2)	void addassign(const sparse &a,double d)	s.addassign(s2,10.0)	Zwiększanie i mnożenie macierzy: baza=baza+a*d
Odejmowanie	sparse spminus(const sparse &a,const sparse &b)	spminus(s1,s2)	Odejmowanie macierzy: a-b
Odejmowanie i mnożenie	sparse spminus(const sparse &a,const sparse &b,double d)	spminus(s1,s2,10.0)	Odejmowanie macierzy: a-b*d
Zmniejszanie	void subassign(const sparse &a)	s.subassign(s2)	Zmniejszanie macierzy: baza=baza-a
Zmniejszanie i mnożenie (wersja 1)	void subassign(double d,const sparse &a)	s.subassign(10.0,s2)	Zmniejszanie macierzy: baza=baza*d-a
Zmniejszanie i mnożenie (wersja 2)	void subassign(const sparse &a,double d)	s.subassign(s2,10.0)	Zmniejszanie macierzy: baza=baza-a*d
Mnożenie	sparse spmul(const sparse &a,const sparse &b)	spmul(s1,s2)	Możenie macierzy po pozycjach: a*b
Dzielenie	sparse spdiv(const sparse &a,const sparse &b)	spdiv(s1,s2)	Dzielenie macierzy po pozycjach: a/b
Przesuwanie elementów			

Przesuwanie wierszy	void shiftrows(int shift)	s.shiftrows(2)	Wiersze macierzy są przesuwane o 'shift'. W przykładzie wiersze są przesuwane o 2 w dół.
Przesuwanie kolumn	void shiftcols(int shift)	s.shifcols(-3)	Kolumny macierzy są przesuwane o 'shift'. W przykładzie kolumny są przesuwane o 3 w lewo.
Operacje z plikami			
Wczytanie	int readtxt(char *name,int show_open_err=1)	s.readtxt("plik.txt")	Wczytanie macierzy rzadkiej z pliku o nazwie 'name'. Parametr 'show_open_err' określa, czy błędy odczytu będą pokazywane. Funkcja „readtxt” zwraca 0, jeśli nie było błędów; i 1, jeśli był błąd odczytu.
Zapisanie	int writetxt(char *name,int show_open_err=1)	s.writetxt("plik.txt")	Zapisanie macierzy rzadkiej do pliku o nazwie 'name'. Parametr 'show_open_err' określa, czy błędy zapisu będą pokazywane. Funkcja „writetxt” zwraca 0, jeśli nie było błędów; i 1, jeśli był błąd zapisu.
Wyświetlanie			
Wyświetlanie	void display(const char *name)	s.display("X")	Macierz rzadka jest wyświetlana w konsoli. Należy podać ‘name’ - nazwę, która ma być wyświetlona.
Listowanie elementów	void list(const char *name)	s.list("X")	Lista elementów macierzy rzadkiej jest wyświetlana w konsoli. Należy podać ‘name’ - nazwę, która ma być wyświetlona.
Pokazanie indeksu	void index(const char *name)	s.index("X")	Indeks macierzy rzadkiej jest wyświetlany w konsoli. Należy podać ‘name’ - nazwę, która ma być wyświetlona.
Inne			
Wybór wiersza	sparse row(const sparse &a,int i)	row(s,2)	Z macierzy 'a' wybierany jest wiersz o numerze 'i'.
Transpozycja	sparse operator ~(const sparse &a) void T()	~s s.T()	Transpozycja macierzy

Porównanie	int compare(const sparse &a,const sparse &b)	compare(s1,s2)	Zwraca wynik porównania macierzy 'a' i 'b'. Jeśli jednakowe, zwraca 1 (true). Jeśli różne, zwraca 0 (false).
Kasowanie	void free()	s.free()	Macierz jest kasowana. Zwalniana jest pamięć, z której korzystała.

1.9 Metody numeryczne (*numproc*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Metody jednowymiarowe			
Całkowanie numeryczne	double integral(double f(double),double a,double b,double tol)	integral(sin,0.0,2.0,1e-6)	Zwraca wynik całkowania funkcji 'f' w przedziale [a,b]. Obliczenia są przeprowadzane z dokładnością 'tol'
Pochodna numeryczna	double derivative(double f(double),double x,double tol)	derivative(cos,0.0,1e-4)	Zwraca pochodną funkcji 'f' w punkcie 'x'. Obliczenia są przeprowadzane z dokładnością 'tol'
Wyszukiwanie minimum funkcji	double fmin(double f(double),double a,double b,double tol)	fmin(fun1,-2.0,2.0,1e-9)	Znajduje i zwraca najmniejszą wartość funkcji 'f' w przedziale [a,b]. Obliczenia są przeprowadzane z dokładnością 'tol'
Wyszukiwanie miejsc zerowych funkcji	double fzero(double f(double),double x,double tol)	fzero(fun2,10.0,1e-12)	Znajduje i zwraca miejsce zerowe funkcji 'f' w pobliżu 'x'. Obliczenia są przeprowadzane z dokładnością 'tol'
Minimalizacja wielowymiarowa			
Wyszukiwanie minimum funkcji wielowymiarowej metodą sympleksów	int fmins(double F(matrix &),matrix &x,double tol)	fmins(FUN1,x,1e-9)	Znajdowane jest minimum funkcji wielowymiarowej F metodą sympleksów. 'x' musi być wektorem kolumnowym o długości zgodnej z długością argumentu funkcji F. Początkowa wartość 'x' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'x'.
Wyszukiwanie minimum funkcji wielowymiarowej metodą gradientów	int fminu(double F(matrix &),double dF(matrix &,int),matrix &x,double tol,EsMethod met=DFP)	fmins(FUN1,dFUN1,x,1e-9)	Znajdowane jest minimum funkcji wielowymiarowej F metodą gradientów. Wymagana jest znajomość pochodnej funkcji dF. 'x' musi być wektorem kolumnowym o długości zgodnej z długością argumentu funkcji F. Początkowa wartość 'x' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'x'. Wybrać można metodę 'met': DFP (Davidon-Fletcher-Powell) lub BFS (Broyden-Fletcher-Shanno).

Wyszukiwanie minimum funkcji wielowymiarowej metodą złożoną	int fminx(double F(matrix &),double dF(matrix &,int),matrix &x,double tol,EsSpeed speed=Normal,EsMethod met=DFP)	fminx(FUN1,dFUN1,x, 1e-9)	Znajdowane jest minimum funkcji wielowymiarowej F metodą złożoną. Wymagana jest znajomość pochodnej funkcji dF. 'x' musi być wektorem kolumnowym o długości zgodnej z długością argumentu funkcji F. Początkowa wartość 'x' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'x'. Wybrać można szybkość estymacji 'speed': Slow, Normal, Fast, Fastest; oraz metodę 'met': DFP (Davidon-Fletcher-Powell) lub BFS (Broyden-Fletcher-Shanno).
Dopasowanie funkcji do danych			
Błąd dopasowania funkcji do danych	double err_fit(const matrix &X,const matrix &Y,double F(double x,matrix &par),matrix &par,EsType es=LSE)	err_fit(X,Y,F,par)	Zwraca błąd dopasowania funkcji F o parametrach 'par' do danych (X - zmienna objaśniająca, Y – zmienna objaśniana). Typ estymacji definiuje parametr 'es', który może przyjmować wartości LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności).
Wywołanie funkcji dla zestawu danych	matrix fun_fit(const matrix &X,double F(double x,matrix &par),matrix &par)	fun_fit(X,F,par)	Zwraca wektor wartości funkcji F dla wektora argumentów 'x' i parametrów 'par'.
Dopasowanie funkcji do danych metodą sympleksów	int fits(const matrix &X,const matrix &Y,double F(double x,matrix &par),matrix &par,double tol,EsType es=LSE)	fits(X,Y,F,par,1e-9,MLE)	Dopasowuje funkcję F o parametrach 'par' do danych (X - zmienna objaśniająca, Y – zmienna objaśniana). Stosowana jest metoda sympleksów. Początkowa wartość 'par' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'par'. Typ estymacji definiuje parametr 'es', który może przyjmować wartości LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności).
Dopasowanie funkcji do danych metodą gradientów	int fitu(const matrix &X,const matrix &Y,double F(double x,matrix &par),double dF(double x,matrix &par,int no_diff),matrix &par,double tol,EsType es=LSE,EsMethod	fitu(X,Y,F,dF,par,1e-9,MLE,BFS)	Dopasowuje funkcję F o parametrach 'par' do danych (X - zmienna objaśniająca, Y – zmienna objaśniana). Stosowana jest metoda gradientów. Wymagana jest znajomość pochodnej funkcji dF. Początkowa wartość 'par' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'par'. Typ estymacji definiuje parametr 'es', który może przyjmować wartości LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności). Wybrać można metodę 'met': DFP (Davidon-Fletcher-Powell) lub BFS (Broyden-Fletcher-Shanno).

	met=DFP)		
Dopasowanie funkcji do danych metodą złożoną	int fitx(const matrix &X,const matrix &Y,double F(double x,matrix &par),double dF(double x,matrix &par,int no_diff),matrix &par,double tol,EsSpeed speed=Normal,EsType es=LSE,EsMethod met=DFP)	fitx(X,Y,F,dF,par,1e-9,MLE,BFS)	Dopasowuje funkcję F o parametrach 'par' do danych (X - zmienna objaśniająca, Y – zmienna objaśniana). Stosowana jest metoda złożona. Wymagana jest znajomość pochodnej funkcji dF. Początkowa wartość 'par' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'par'. Wybrać można szybkość estymacji 'speed': Slow, Normal, Fast, Fastest; typ estymacji: LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności); oraz metodę 'met': DFP (Davidon-Fletcher-Powell) lub BFS (Broyden-Fletcher-Shanno).
Dopasowanie funkcji do danych wielowymiarowych			
Błąd dopasowania funkcji wielowymiarowej do danych	double err_fit(const matrix &X,const matrix &Y,double F(matrix &x,matrix &par),matrix &par,EsType es=LSE)	err_fit(X,Y,F,par)	Zwraca błąd dopasowania wielowymiarowej funkcji F o parametrach 'par' do danych (X - zmienne objaśniające, Y – zmienna objaśniana). Typ estymacji definiuje parametr 'es', który może przyjmować wartości LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności).
Wywołanie funkcji wielowymiarowej dla zestawu danych	matrix fun_fit(const matrix &X,double F(matrix &x,matrix &par),matrix &par)	fun_fit(X,F,par)	Zwraca wektor wartości wielowymiarowej funkcji F dla wektora argumentów 'x' i parametrów 'par'.
Dopasowanie funkcji wielowymiarowej do danych metodą sympleksów	int fits(const matrix &X,const matrix &Y,double F(matrix &x,matrix &par),matrix &par,double tol,EsType es=LSE)	fits(X,Y,F,par,1e-9,MLE)	Dopasowuje wielowymiarową funkcję F o parametrach 'par' do danych (X - zmienne objaśniające, Y – zmienna objaśniana). Stosowana jest metoda sympleksów. Początkowa wartość 'par' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'par'. Typ estymacji definiuje parametr 'es', który może przyjmować wartości LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności).
Dopasowanie funkcji wielowymiarowej do	int fitu(const matrix &X,const matrix &Y,double	fitu(X,Y,F,dF,par,1e-9,MLE,BFS)	Dopasowuje wielowymiarową funkcję F o parametrach 'par' do danych (X - zmienne objaśniające, Y – zmienna objaśniana). Stosowana jest metoda gradientów. Wymagana jest znajomość pochodnej funkcji dF.

danych metodą gradientów	F(matrix &x,matrix &par),double dF(matrix &x,matrix &par,int no_diff),matrix &par,double tol,EsType es=LSE,EsMethod met=DFP)		Początkowa wartość 'par' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'par'. Typ estymacji definiuje parametr 'es', który może przyjmować wartości LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności). Wybrać można metodę 'met': DFP (Davidon-Fletcher-Powell) lub BFS (Broyden-Fletcher-Shanno).
Dopasowanie funkcji wielowymiarowej do danych metodą złożoną	int fitx(const matrix &X,const matrix &Y,double F(matrix &x,matrix &par),double dF(matrix &x,matrix &par,int no_diff),matrix &par,double tol,EsSpeed speed=Normal,EsType es=LSE,EsMethod met=DFP)	fitx(X,Y,F,dF,par,1e-9,MLE,BFS)	Dopasowuje wielowymiarową funkcję F o parametrach 'par' do danych (X - zmienne objaśniające, Y – zmienna objaśniana). Stosowana jest metoda złożona. Wymagana jest znajomość pochodnej funkcji dF. Początkowa wartość 'par' powinna znajdować się blisko rozwiązania. Obliczenia są przeprowadzane z dokładnością 'tol', a wynik jest zwracany w zmiennej 'par'. Wybrać można szybkość estymacji 'speed': Slow, Normal, Fast, Fastest; typ estymacji: LSE (najmniejszych kwadratów) lub MLE (największej wiarygodności); oraz metodę 'met': DFP (Davidon-Fletcher-Powell) lub BFS (Broyden-Fletcher-Shanno).

1.10 Sieć neuronowa (*neural_net*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Deklaracja sieci neuronowej	neural_net()	neural_net nn	Zadeklarowanie sieci neuronowej. W przykładzie sieć ma nazwę 'nn'
Tworzenie sieci o zadanej architekturze warstw	neural_net(int i,int h,int ha,int oa)	neural_net nn(1,2,logistic,linear)	Zadeklarowanie sieci neuronowej, posiadającej 'i' wejść, 'h' neuronów w warstwie ukrytej. Aktywacja warstwy ukrytej odbywa się za pomocą funkcji 'ha', a aktywacja warstwy wyjściowej za pomocą funkcji 'oa'. Numery i nazwy funkcji aktywacji: 0 – undefined (nieokreślona) 1 – logistic ($\frac{1}{1+\exp(-x)}$) 2 – tanhyp ($\tanh(x)$) 3 – linear (x) W przykładzie powstaje sieć o jednym wejściu, 2 neuronach ukrytych. Funkcje aktywacji są typu 'logistic'.
Definiowanie wzorców			
Definiowanie wzorców (danych)	void set_patterns(const matrix &p_in,const matrix &p_out,int set=0)	nn.set_patterns(x,y,learn_set)	Definiowane są wzorce uczące: 'p_in' - wejściowe (zmienne objaśniające), 'p_out' – wyjściowe (zmienna objaśniana). Wybrać należy zbiór 'set': 0 – learn_set (zbiór uczący) 1 – valid_set (zbiór walidacyjny, pomocniczy przy uczeniu) 2 – test_set (niezależny zbiór testowy) W przykładzie definiowany jest zbiór uczący dla zmiennych x i zmiennej objaśnianej y.
Liczba wzorców	int patterns(int set=0)	nn.patterns(learn_set)	Liczba wzorców w danych zbiorze 'set'.
Informacja o danych	void datainfo()	nn.datainfo()	Zbiorcza informacja o danych: liczba wzorców we wszystkich zbiorach danych (0, 1 i 2)

Inicjalizacja			
Wybór funkcji aktywacji	void set_activ_fun(int ha=logistic,int oa=logistic)	nn.set_activ_fun(logistic,linear)	Wybierane są funkcje aktywacji dla warstwy ukrytej i wyjściowej.
Inicjalizacja	void init(double c=0.02)	nn.init()	Inicjalizacja połączeń przed rozpoczęciem uczenia. Parametr 'c' określa wielkość parametrów losowych.
Prosta inicjalizacja	void init_simply(double c=0.02)	nn.init_simply()	Prosta inicjalizacja połączeń przed rozpoczęciem uczenia. Inicjalizowane jest tylko po jednym połączeniu od każdego wejścia. Parametr 'c' określa wielkość parametrów losowych.
Inicjalizacja pojedynczego wejścia	void init_single_input(int v_in,int h,int pos=1,double c=0.02)	nn.init_single_input(10,5,1)	Inicjalizacja połączeń od danego wejścia. Wybiera się numer wejścia 'v_in', liczbę aktywowanych neuronów ukrytych 'h' i położenie w warstwie ukrytej 'pos'. Parametr 'c' określa wielkość parametrów losowych. W przykładzie inicjalizowane są połączenia od 10 wejścia do 1, 2, 3, 4 i 5 neuronu ukrytego.
Inicjalizacja do realizacji funkcji binarnej	void init_binfun(int v_in,int pos=1)	nn.init_binfun(10,15)	Inicjalizacja realizująca kodowanie binarne. Dotyczy tylko sieci aktywowanych za pomocą funkcji (tanhyp, linear) i zmiennej zero-jedynkowej na wybranym wejściu. 'v_in' to numer wejścia, 'pos' to pozycja wykorzystanego neuronu w warstwie ukrytej (wykorzystany jest 1 neuron). Połączenia są ustawiane na wartości realizujące kodowanie binarne (nie wymagają uczenia). W przykładzie inicjalizuje się połączenia dla zmiennej nr 10, biegnące do neuronu 15.
Inicjalizacja do realizacji kodowania ciągłego	void init_confun(int v_in,int pos=1)	nn.init_confun(10,15)	Inicjalizacja realizująca kodowanie ciągłe funkcją tanh4. Dotyczy tylko sieci aktywowanych za pomocą funkcji (tanhyp, linear) i zmiennej ciągłej na wybranym wejściu. 'v_in' to numer wejścia, 'pos' to pozycja wykorzystanego neuronu w warstwie ukrytej (wykorzystany jest 1 neuron). Połączenia są ustawiane na wartości realizujące kodowanie ciągłe tanh (wymagane jest uczenia). W przykładzie inicjalizuje się połączenia dla zmiennej nr 10, biegnące do neuronu 15.
Podgląd struktury			
Liczba warstw wejściowych	int size_in()	nn.size_in()	Zwraca liczbę neuronów wejściowych
Liczba warstw ukrytych	int size_hid()	nn.size_hid()	Zwraca liczbę neuronów ukrytych
Funkcja aktywująca	int act_hid()	nn.act_hid()	Zwraca numer funkcji aktywującej warstwę ukrytą:

warstwy ukryte			0 – undefined 1 – logistic 2 – tanhyp 3 – linear
Funkcja aktywująca warstwę wyjściową	int act_out()	nn.act_out()	Zwraca numer funkcji aktywującej warstwę wyjściową: 0 – undefined 1 – logistic 2 – tanhyp 3 – linear
Struktura sieci	void structure()	nn.structure()	Wyświetla strukturę sieci: - layer size – liczba neuronów w warstwie wejściowej, ukrytej i wyjściowej - activation – funkcje aktywacji (wejście → ukryte, ukryte → wyjście) - parameters – liczbę niezerowych parametrów dla połączeń (wejście → ukryte, bias → ukryte, ukryte → wyjście, bias → wyjście)
Manipulacja strukturą			
Zmiana rozmiaru sieci	void resizec(int i_new,int h_new)	nn.resizec(10,5)	Zmiana liczby wejść z bieżącej na 'i_new' oraz liczby neuronów ukrytych z bieżącej na 'h_new'. Zachowywane są wartości połączeń.
Dodanie neuronu w warstwie ukrytej	void add_hid(int v_in,int pos,double c=0.02)	nn.add_hid(8,5)	Dodanie neuronu w warstwie ukrytej. Połączenie od wejścia 'v_in' do pozycji 'pos'. Parametr 'c' określa wielkość parametrów losowych.
Liczba używanych neuronów ukrytych	int used_hid()	nn.used_hid()	Zwraca liczbę używanych neuronów w warstwie ukrytej. Neuron jest używany, jeśli jest aktywowany przez wcześniejszą warstwę.
Przesunięcie neuronów w warstwie ukrytej	void shift_hid(int s)	nn.shift_hid(10)	Neurony w warstwie ukrytej są przesuwane o 's' pozycji.
Łączenie dwóch sieci	void merge(neural_net &n1,neural_net &n2,double c1=0.5,double c2=0.5,double c0=0.0)	nn.merge(n1,n2)	Łączenie sieci 'n1' i 'n2'. Otrzymana sieć daje wyniki będące kombinacją liniową składowych: value(x)=c1*n1.value(x)+c2*n2.value(x)+c0

Dołączenie sieci do istniejącej sieci	void join(neural_net &n1,double c1=1.0)	nn.join(n1)	Dołączenie sieci 'n1' do bieżącej sieci. Po dołączeniu bieżąca sieć daje wyniki będące kombinacją liniową sieci przed dołączeniem i sieci dołączanej: value(x)=value(x)+c1*n1.value(x)
Przemnożenie wag drugiej warstwy przez stałą	void w2b2_mul(double c)	nn.w2b2_mul(2.0)	Połączenia od warstwy ukrytej do wyjściowej są mnożone przez stałą 'c'. Przykładowo, sieć będzie dawać wyniki 2 razy większe.
Dodanie wartości do biasu drugiej warstwy	void b2_plus(double c)	nn.b2_plus(8.5)	Bias warstwy wyjściowej jest zwiększany o stałą 'c'. Przykładowo, sieć będzie dawać wyniki zwiększone o 8.5.
Wypełnienie nieużywanych połączeń losowymi wartościami	void refill(double c)	nn.refill(0.02)	Nieużywane połączenia są inicjowane losowymi wartościami, o których wielkości decyduje stała 'c'.
Losowa przebudowa struktury			
Dodanie jednego połączenia w pierwszej warstwie	void rand_add_w1(double c)	nn.rand_add_w1(0.02)	Dodawane jest losowo połączenie w pierwszej warstwie
Dodanie jednego połączenia w drugiej warstwie	void rand_add_w2(double c)	nn.rand_add_w2(0.02)	Dodawane jest losowo połączenie w drugiej warstwie
Dodanie połączenia w obu warstwach	void rand_add_w2w1(double c)	nn.rand_add_w2w1(0.02)	Dodawane jest losowo połączenie w pierwszej i w drugiej warstwie
Usunięcie jednego połączenia w pierwszej warstwie	void rand_del_w1()	nn.rand_del_w1()	Usuwane jest losowo połączenie w pierwszej warstwie
Usunięcie jednego połączenia w drugiej	void rand_del_w2()	nn.rand_del_w2()	Usuwane jest losowo połączenie w drugiej warstwie

warstwie			
Redukcja połączeń sieci	void reduction(double c=0.02)	nn.reduction(0.01)	Połączenia sieci o wadze mniejszej co do wartości bezwzględnej od 'c' są zerowane
Aktywacja połączeń nieaktywnych	void rand_fill0(double c=0.02)	nn.rand_fill0(0.01)	Połączenia o wartości zero są aktywowane wartościami losowymi z zakresu [0,c]
Dopasowanie sieci do danych			
Dopasowanie sieci do danych	int fit(int iter=100,int grad_desc=1,int lev_marq=1,int layer_1=1)	nn.fit()	Sieć jest dopasowywana do danych. Znajdowane jest minimum funkcji błędu dla prób bootstrapowych. Parametry funkcji: - iter – liczba iteracji (domyślnie 100) - grad_desc – metoda gradientu sprzężonego (domyślnie włączona) - lev_marq – metoda Levenberga-Marquardta (domyślnie włączona) - layer_1 – modyfikacja połączeń w pierwszej warstwie (domyślnie włączona)
Uczenie sieci	int learn(int min_iter=100,int max_iter=1000,int slow_steps=10,double tol_err=1e-8,int grad_desc=1,int lev_marq=1,int layer_1=1,int regul=1)	nn.learn()	Sieć jest uczona. W wyniku otrzymywany jest model o właściwościach generalizujących. Parametry funkcji: - min_iter – minimalna liczba iteracji (domyślnie 100) - max_iter – maksymalna liczba iteracji (domyślnie 1000) - slow_steps – liczba wolnych kroków, po których uczenie zostaje przerwane (domyślnie 10) - tol_err – dokładność uczenia (domyślnie 1e-8) - grad_desc – metoda gradientu sprzężonego (domyślnie włączona) - lev_marq – metoda Levenberga-Marquardta (domyślnie włączona) - layer_1 – modyfikacja połączeń w pierwszej warstwie (domyślnie włączona) - regul – czy ma być zastosowana regularyzacja (domyślnie tak)

Uczenie z równoczesną walidacją	int learn_and_valid(int min_iter=100,int max_iter=1000,int slow_steps=10,double tol_err=1e-8,int grad_desc=1,int lev_marq=1,int layer_1=1)	nn.learn_and_valid()	Sieć jest uczona i jednocześnie walidowana. W wyniku otrzymywany jest model o właściwościach generalizujących. Parametry funkcji są analogiczne jak dla 'learn', z tym że nie ma parametru regularyzacji. (Metoda niepolecana, ponieważ walidacja niedostatecznie zabezpiecza przed przeuczeniem.)
Resetowanie procesu uczenia	void reset()	nn.reset()	Proces uczenia jest resetowany
Miary dopasowania			
Błąd dopasowania	double error(int set=0)	nn.error(0)	Zwraca błąd dopasowania dla zbioru o numerze 'set'
Korelacja wartości obserwowanych i predykowanych	double corr(int set=0)	nn.corr(0)	Zwraca wartość korelacji wartości obserwowanych i predykowanych dla zbioru o numerze 'set'
Skuteczność predykcji	double AR(int set=0)	nn.AR(0)	Zwraca wskaźnik AR dla zbioru o numerze 'set'
Obliczanie wartości			
Wartość predykcji	double value(const sparse &x)	nn.value(x)	Zwraca wartość predykcji sieci dla wejścia 'x'
Wartość predykcji dla wzorców	matrix values(int set=0)	nn.values(0)	Zwraca zestaw wartości predykcji sieci dla zbioru wzorców o numerze 'set'
Zapis i odczyt			
Zapis sieci	void save(Text name)	nn.save("siec.txt")	Zapis sieci do pliku o nazwie 'name'
Wczytanie sieci	void load(Text name)	nn.load("siec.txt")	Wczytanie sieci z pliku o nazwie 'name'

Wczytanie struktury sieci	void load_structure(Text name)	nn.load_structure("siec.txt")	Wczytanie struktury sieci z pliku o nazwie 'name'. Połączenia są inicjowane losowymi wartościami.
Skasowanie sieci	void free()	nn.free()	Skasowanie sieci. Zwalniana jest pamięć przydzielona na ten obiekt.

1.11 Wykres (*gplot*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Deklaracja obiektu wykres	gplot()	gplot gp	Deklarowana jest zmienna typu wykres
Deklaracja obiektu wykres i otworzenie pliku wykresu do edycji	gplot(Text name,Text path=Text())	gplot gp("sample")	Deklarowana jest zmienna typu wykres i otwierany jest plik o podanej nazwie do edycji (rysowania). Opcjonalnie można podać ścieżkę do katalogu, w którym wykres będzie zapisany. Domyślnie wybierany jest bieżący katalog. Po tej instrukcji można rozpocząć rysowanie.
Otworzenie pliku wykresu do edycji	void open(Text name,Text path=Text())	gp.open("sample2")	Otwierany jest plik wykresu. Po tej instrukcji można rozpocząć rysowanie.
Zamknięcie pliku wykresu	void close()	gp.close()	Zamykany jest plik wykresu. Ta instrukcja kończy rysowanie.
Właściwości obiektu wykres			
Nazwa pliku wykresu	Text getname()	gp.getname()	Zwracana jest nazwa pliku wykresu
Sprawdzenie czy plik jest otwarty do edycji	int isopen()	gp.isopen()	Zwracana jest wartość 1 (true), jeśli otwarto plik do edycji. Albo 0 (false) - w przeciwnym przypadku.
Parametry wykresu			
Wykres wielokrotny	void multiplot(int set=1,int rows=1,int cols=2)	gp.multiplot(1,2,2)	Włączenie/wyłączenie trybu wielu wykresów (‘set’ równe 1 oznacza włączenie, a równe 0 wyłączenie). Można wybrać liczbę wykresów w pionie (rows) i poziomie (cols). W podanym przykładzie włączony zostaje tryb z czterema wykresami na jednej ilustracji (2x2).
Tytuł	void title(Text t)	gp.title("Tytuł	Wykres jest tytułowany

		wykresu")	
Styl	void style(Text t)	gp.style("fill solid border -1")	Ustawienie stylu rysowania
Szerokość słupka	void boxwidth(double d)	gp.boxwidth(5)	Ustawienie szerokości słupków dla wykresów, które zawierają ten element.
Zakresy osi	void xrange(double d1,double d2) void yrange(double d1,double d2) void zrange(double d1,double d2)	gp.xrange(-10,10)	Wybór zakresu dla osi X, Y i Z
Tytuły osi	void xlabel(Text t) void ylabel(Text t) void zlabel(Text t)	gp.xlabel("Oś X")	Dodawanie podpisów osi X, Y i Z
Legenda	void key(Text t)	gp.key("bot right")	Włączenie legendy i wybór jej położenia
Próbkowanie	void samples(double d)	gp.samples(100)	Liczba próbek rysowanej funkcji
Próbkowanie ISO	void isosamples(double d)	gp.isosamples(10)	Gęstość siatki dla wykresów 3D
Tryb parametryczny	void parametric(int set)	gp.parametric(1)	Włączenie/wyłączenie rysowania funkcji parametrycznych (‘set’ równe 1 oznacza włączenie, a równe 0 wyłączenie)
Kąt widzenia	void view(double d1,double d2)	gp.view(65,55)	Wybór kąta widzenia dla wykresów 3D
Pojedyncza etykieta	void label(int i,Text t,double d1,double d2)	gp.label(1,"Jakiś napis”,200,100)	Dodaje dowolny napis w miejscu o współrzędnych (d1,d2). Parametr ‘i’ umożliwia numerowanie etykiet.

Obiekt	void object(Text t)	gp.object("polygon from 10,10 to 20,20 fc rgb 'orange'")	Umożliwia rysowanie obiektów (prostokątów, kół, elips, wielokątów)
Typ znaczników osi	void xtics(Text t) void ytics(Text t) void ztics(Text t)	gp.xtics("rotate by -90")	Umożliwia ustawienie sposobu wyświetlania znaczników osi X, Y i Z.
Margines	void bmargin(double d)	gp.bmargin(10)	Ustawienie szerokości dolnego marginesu
Reset	void reset()	gp.reset()	Zresetowanie wykresu. Przywracane są wszystkie standardowe ustawienia.
Rysowanie wykresu			
Wykres danych z pliku lub funkcji zdefiniowanej napisem	void plot(Text data1,Text modif1=Text(),Text title1=Text(),Text style1=Text(),…)	gp.plot("dane_wykresu.txt","1:2","Zmienna 1","boxes", "~","1:3","Zmienna 2","boxes") gp.plot("2.5*sin(x)+x/10.0","","Wykres funkcji")	Rysowanie wykresu 2D ma podstawie danych z pliku tekstowego lub funkcji zdefiniowanej napisem. Parametry: - data1 – nazwa pliku lub funkcja - modif1 – wybór zakresu kolumn - title1 – tytuł krzywej - style1 – styl rysowania krzywej Można zdefiniować do pięciu niezależnych źródeł danych lub funkcji, posiadających identyczny zestaw parametrów jak opisane wyżej. Istnieje również funkcja ‘splot’, mająca identyczne parametry, służąca do rysowania powierzchni (wykres 3D). W pierwszym przykładzie rysunek korzysta z pliku „dane_wykresu.txt”. Rysowany jest wykres pudełkowy dla danych x z kolumny 1 i danych y z kolumny 2. Krzywa otrzymuje opis „Zmienna 1”. Rysowana jest także druga krzywa na podstawie danych z tego samego pliku (znak ~ oznacza powtórzenie źródła danych). Dane x

			są brane z kolumny 1, a dane y z kolumny 3. Krzywa otrzymuje opis „Zmienna 2”. W drugim przykładzie rysowana jest funkcja 2.5*sin(x)+x/10. Krzywa otrzymuje opis „Wykres funkcji”.
Wykres danych z tabeli	void plot(table tab1,Text modif1=Text(),Text title1=Text(),Text style1=Text(),…)	gp.plot(tabela,"1:2","Zmienna 1","boxes","~","1:3","Zmienna 2","boxes")	Rysowanie wykresu 2D na podstawie danych z podanej tabeli. Funkcjonalność identyczna jak funkcji ‘plot’ rysującej na podstawie danych z pliku tekstowego. ‘Splot’ tworzy wykres 3D.
Wykres danych z macierzy	void plot(matrix mat1,Text modif1=Text(),Text title1=Text(),Text style1=Text(),…)	gp.plot(macierz,"1:2","Zmienna 1","boxes","~","1:3","Zmienna 2","boxes")	Rysowanie wykresu 2D na podstawie danych z podanej macierzy. Funkcjonalność identyczna jak funkcji ‘plot’ rysującej na podstawie danych z pliku tekstowego. ‘Splot’ tworzy wykres 3D.
Inne instrukcje			
Ustawienie	void set(Text t)	gp.set("key bot right\n")	Umożliwa ustawienie dowolnego parametru wykresu
Definicja	void define(Text t)	gp.define ("min(a,b)=(a<=b)*a+(a>b)*b")	Umożliwia zdefiniowanie funkcji, wykorzystywanej na wykresie.
Instrukcja	void inst(Text t)	gp.inst("set xdata time")	Wywołanie dowolnej instrukcji gnuplot.

Instrukcja otwarta	void instc(Text t)	gp.instc("plot ‘dane_wykresu.txt’ 1:2,")	Wywołanie dowolnej instrukcji gnuplot bez kończenia wiersza. Instrukcja otwarta umożliwia kontynuowanie instrukcji gnuplot w następnej linii programu.
Odstęp	void sep(char c='-',int l=70)	gp.sep(‘*’,30)	Dodanie separatora do kodu skryptu
Koniec linii	void endl(int n=1)	gp.endl()	Znak końca wiersza
Komentarz	void comment(Text t)	gp.comment("Jakiś komentarz")	Komentarz skryptu gnuplot
Komentarz specjalny	void speccom(Text t,char c='-',int l=70)	gp.speccom("Jakiś komentarz")	Komentarz skryptu wyróżniony liniami separującymi
Pauza	void pause(int time=-1,Text info="Hit return to continue")	gp.pause()	Zatrzymuje skrypt gnuplot po wyświetleniu wykresu. Można wybrać czas (-1 oznacza oczekiwanie na kliknięcie myszką). W okienku informacyjnym wyświetlany jest komunikat zdefiniowany przez parametr ‘info’.

1.12 Raport HTML (*html*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Deklaracja obiektu raport	html(Text title="HL++ html",Text description="Html created by HL++",Text keywords="HL++")	html raport	Deklarowana jest obiekt typu raport HTML.
Ustawienie koloru tła	void background(Text color)	raport.backgroundcolor("blue")	Ustawiane jest tło strony.
Dodanie zakładki	void tab(Text name)	raport.tab("Pierwsza zakładka")	Dodawanie nowej zakładki do raportu. Raport musi mieć co najmniej jedną zakładkę. Jeżeli jest więcej niż 1 zakładka, to u góry strony automatycznie zostanie wygenerowany pasek nawigacji między zakładkami.
Właściwości tekstu			

Klasa właściwości tekstu	class text_properties
	{
	public:
	int bold,
	italic,
	underline,
	font_size; znakow
	Text font_color,
	font_face;
	
	text_properties();
	};
		text_properties wlasciwosci_tekstu	Klasa definująca właściwości tekstu umieszczanego w raporcie za pomocą metody add_text. Każdy umieszczany tekst ma własne właściwości, takie jak:
			- pogrubienie (bold)
			- kursywa (italic)
			- podkreślenie (underline)
			- wielkość znaków (font_size)
			- kolor znaków (font_color)
			- rodzaj czcionki (font_face)
			Po utworzeniu zmiennej text_properties te właściwości przyjmują wartości domyślne (zwykły tekst), ale

			można je zmieniać jak w przykładzie poniżej:
			
			text_properties wlasciwosci_tekstu;
			
			wlasciwosci_tekstu.bold=1;
			
			wlasciwosci_tekstu.size=20;

Właściwości tabeli			

Klasa właściwości tabeli	class table_properties
	{
	public:
	Text captition,
	captition_align,
	width,
	height,
	align,
	bgcolor,
	names_color;
	int col_names,
	row_names,
	border,
	cellpadding,
	cellspacing;
	
	table_properties();
	};
		table_properties wlasciwosci_tabeli	Klasa definująca właściwości tabel umieszczanych w raporcie za pomocą metody add_table. Każda umieszczana tabela ma własne właściwości, takie jak:
			
			- tytuł (captition)
			
			- położenie tytułu (captition_align)
			
			- szerokość w pikselach lub % (width)
			
			- wysokość w pikselach lub % (height)
			
			- wyrównanie względem tekstu (align)
			
			- kolor tła (bgcolor)
			
			- kolor wiersza z nazwami kolumn (names_color)
			
			- wybór czy pokazywać nazwy kolumn (col_names)
			
			- wybór czy pokazywać nazwy wierszy (row_names)
			
			- grubość zewnętrznej ramki w pikselach (border)
			
			- szerokość marginesów poziomych i pionowych (cellpadding)
			
			- szerokość odstępu między sąsiednimi komórkami (cellspacing)
			
			Po utworzeniu zmiennej table_properties te właściwości przyjmują wartości domyślne, ale można je zmieniać jak w przykładzie poniżej:
			
			table_properties wlasciwosci_tabeli;

			wlasciwosci_tabeli.captition="Tabela 1. Przykładowa"; wlasciwosci_tabeli.row_names=0;
Dodawanie treści			
Dodanie rozdziału	void chapter(Text name,const text_properties &p)	raport.chapter("Rozdział 1", wlasciwosci_tekstu)	Do raportu dodawany jest nowy rozdział. Tytuł rozdziału ma podane wlasciwosci_tekstu. Dodawanie rozdziałów powoduje automatyczne utworzenie spisu treści na pasku nawigacyjnym z lewej strony raportu.
Dodanie tekstu	void htext(Text t,const text_properties &p)	raport.htext("Jakiś napis",wlasciwosci_tekstu)	Do raportu dodawany jest tekst o podanych właściwościach.
Dodanie paragrafu	void paragraph(Text t,const text_properties &p,Text align="left")	raport.paragraph("Jakiś napis",wlasciwosci_tekstu)	Do raportu dodawany jest tekst w paragrafie o podanych właściwościach. Po kliknięciu w paragraf tekst może być dynamicznie modyfikowany.
Dodanie ilustracji	void image(Text name,Text dir="",Text align="left",int width=600,int height=400)	raport.image("obraze k.jpg")	Do raportu dodawana jest ilustracja (obraz).
Dodanie odwołania	void href(Text name,Text dir="",Text description="reference",Text image="")	raport.href("rozdzial2. html")	Do raportu dodawane jest odwołanie wewnętrzne lub zewnętrzne.
Dodanie tabeli	void htable(const table &tab,const table_properties &p) void htable(const table &tab,const matrix &conv,const	raport.htable(tabela, wlasciwosci_tabeli) raport.htable(tabela,c (2,2,0,1),wlasciwosci_ tabeli)	Do raportu dodawana jest tabela o podanych właściwościach. Opcjonalny parametr 'conv' definiuje sposób prezentacji liczb. Jest to wektor o długości odpowiadającej liczbie kolumn w tabeli. Każda wartości (liczba całkowita) definiuje sposób prezentacji w kolejnych kolumnach. Liczby dodanie wskazują, że liczby w kolumnie mają być wyświetlone jako liczby rzeczywiste o zadanej dokładności po przecinku. Wartość 0 wskazuje, że mają być wyświetlane liczby całkowite. Wartości ujemne wskazują, że mają być wyświetlane wartości procentowe o zadanej dokładności po przecinku.

	table_properties &p)		
Dodanie macierzy	void hmatrix(const matrix &mat,const matrix &conv,const table_properties &p)	raport.hmatrix(macierz,c(2,3,6),wlasciwosci_tabeli)	Do raportu dodawana jest macierz. Parametr 'conv' definiuje sposób prezentacji liczb (tak samo jak dla funkcji add_table).
Dodanie JavaScriptu	void js(Text name,Text dir="",int width=600,int height=400)	raport.js("wykres_gplot.js")	Do raportu dodawany jest dowolny JavaScript. JavaScriptem może być wykres utworzony za pomocą gplot, jeżeli ustawi się właściwości pliku wynikowego: set terminal canvas.
Dodanie separatora	void endline(int n=1)	raport.endline()	Do raportu dodawane są linie odstępu.
Inne instrukcje			
Czyszczenie raportu	void clear()	raport.clear()	Obiekt raport jest czyszczony. Powstaje pusty raport.
Zapisanie raportu	void save(Text dir,Text name)	raport.save("","raport.html")	Raport zapisywane jest na dysku w zadanej lokalizacji i pod określoną nazwą.

1.13 Symulacje i kwantyle (*simulation, quantiles*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania
Deklaracja obiektu symulacje	simulation()	simulation s	Tworzony jest pusty obiekt symulacje.
Deklaracja obiektu z parametrami histogramu	simulation(double hist_min,double hist_max,int his_num=20)	simulation s(-4.0,4.0,50)	Tworzony jest pusty obiekt symulacje. Zdefiniowane zostają parametry histogramu: minimalna wartość (hist_min), maksymalna wartość (hist_max) i liczba przedziałów (hist_num). Domyślnie jest 20 przedziałów histogramu.
Resetowanie symulacji	void reset()	s.reset()	Resetowanie symulacji. Obiekt symulacji jest przywracany do stanu początkowego, gdy liczba obserwacji wynosi 0.
Dodawanie obserwacji	void add(double e)	s.add(normal_sample(0.0,1.0,seed))	Do symulacji dodawana jest jedna obserwacja. W podanym przykładzie jest to liczba z rozkładu normalnego N(0,1). Funkcje losujące liczby z różnych rozkładów są dostępne w biblitece prob.hpp. Punkt startowy generatora losowego (tzw. ziarno) można zadeklarować tak: int seed[1]={10}; lub int seed[1]={time(0)}; W pierwszym przypadku ziarno jest deterministyczne (równe 10), w drugim przypadku zależy od czasu uruchomienia programu.
Wczytanie obiektu	void load(Text name)	s.load("symul.dat")	Obiekt symulacje jest wczytywany z pliku o podanej nazwie.
Zapisanie obiektu	void save(Text name)	s.save("symul.dat")	Obiekt symulacje jest zapisywany do pliku o podanej nazwie.
Kumulowanie symulacji	void operator +=(const simulation &s)	s+=s2;	Symulacje z dwóch obiektów są sumowane w pierwszym z nich. W podanym przykładzie do obiektu 's' są dodawane wszystkie symulacje zgromadzone w obiekcie 's2'.

Wyniki symulacji			
Liczba symulacji	long length()	s.length()	Zwracana jest liczna wykonanych symulacji.
Najmniejsza wartość	double min()	s.min()	Zwracana jest wartość najmniejsza.
Największa wartość	double max()	s.max()	Zwracana jest wartość największa.
Zakres	double range()	s.range()	Zwracany jest zakres wartości (tj. maksimum - minimum).
Suma	double sum()	s.sum()	Zwracana jest suma wartości.
Średnia arytmetyczna	double mean()	s.mean()	Zwracana jest średnia arytmetyczna.
Średnia geometryczna	double gmean()	s.gmean()	Zwracana jest średnia geometryczna wartości dodatnich.
Średnia harmoniczna	double hmean()	s.hmean()	Zwracana jest średnia harmoniczna wartości różnych od 0.
Dominanta	double mode()	s.mode()	Zwracana jest wartość najczęstsza.
Wariancja	double var()	s.var()	Zwracana jest variancja rozkładu.
Odchylenie standardowe	double stdev()	s.stdev()	Zwracane jest odchylenie standardowe.
Skośność	double skewness()	s.skewness()	Zwracana jest skośność.
Kurtoza	double kurtosis()	s.kurtosis()	Zwracana jest kurtoza.
Histogram	matrix hist()	s.hist()	Zwracana jest macierz z histogramem rozkładu.

Obliczanie kwantyli:

Metoda	Deklaracja metody	Przykład wywołania	Opis działania

Deklaracja obiektu kwantyle	quantiles(double p=0.5)	quantiles q(0.3)	Deklarowana jest obiekt kwantyle. Obliczany będzie kwantyl 'p' (domyślnie kwantyl 0.5, czyli mediana). W podanym przykładzie obiekt będzie obliczał kwantyl 0.3.
Resetowanie obiektu	void reset(double p)	q.reset(0.4)	Resetowanie symulacji służących do wyliczenia kwantyla. Obiekt 'kwantyle' jest przywracany do stanu początkowego, gdy liczba obserwacji wynosi 0. Wymagane jest podanie, jaki kwantyl będzie następnie obliczany (parametr 'p').
Dodawanie obserwacji	void add(double e)	q.add(normal_sample(0.0,1.0,seed))	Do symulacji dodawana jest jedna obserwacja. W podanym przykładzie jest to liczba z rozkładu normalnego N(0,1). Więcej informacji patrz: funkcja add w obiekcie „simulation”.
Wynik	double result()	q.result()	Obliczany jest kwantyl rozkładu.

Dystrybuanta	double cdf(double x)	q.cdf(0.0)	Obliczana jest wartość dystrybuanty w zadanym punkcie położonym w pobliżu wyznaczonego kwantyla. Obiekt kwantyle nie gromadzi wszystkich obserwacji, a tylko te w pobliżu szukanego wyniku. Jeżeli wartość x będzie odległa od wyniku, to pojawi się komunikat ostrzegawczy, że wartości dystrybuanty nie można wyznaczyć. Ta funkcja może służyć np. do sprawdzenia, czy wynik otrzymany funkcją result() jest prawidłowy:
			
			int seed[1]={1};
			
			quantiles q(0.5);
			for (long n=1; n<=1000000; n++) q.add(normal_sample(0.0,1.0,seed));
			
			double mediana=q.result();
			cout<<"mediana="<<mediana<<endl;
			cout<<"cdf(mediana)="<<q.cdf(mediana)<<endl;
			
			Oczekiwany wynik działania programu przy dużej liczbie symulacji to:
			
			mediana=0.0
			cdf(mediana)=0.5
			
			Funkcja jest wykorzystywana przez obiekt „quantiles2”, kumulujący obiekty „quantiles” (patrz poniżej).

Wczytanie obiektu	void load(Text name)	q.load("kwantyl1.dat")	Obiekt kwantyle jest wczytywany z pliku o podanej nazwie.

Zapisanie obiektu	void save(Text name)	q.save("kwantyl1.dat")	Obiekt kwantyle jest zapisywany do pliku o podanej nazwie.
Kumulowanie obiektów kwantylowych			
Deklaracja obiektu sumującego zestawy symulacji	quantiles2(int max_size=10)	quantiles2 qq(2)	Tworzony jest obiekt, w którym można zsumować, wiele obiektów kwantylowych (typu quantiles). Parametr max_size określa, ile maksymalnie obiektów będzie zsumowanych.
Resetowanie obiektu	void reset(int max_size=10)	qq.reset()	Resetowanie obiektu. Obiekt 'kwantyle2' jest przywracany do stanu początkowego, gdy nie zawiera, żadnych danych. Można zmienić maksymalną liczbę obiektów składowych (parametr 'max_size').
Kumulowanie symulacji	void operator +=(const quantiles &q)	qq+=q	Symulacje z obiektów „kwantyle" są sumowane w obiekcie „kwantyle2". W podanym przykładzie do obiektu 'qq' są dodawane wszystkie symulacje zgromadzone w obiekcie 'q'.
Wynik	double result()	qq.result()	Obliczany jest kwantyl rozkładu.
Dystrybuanta	double cdf(double x)	qq.cdf()	Obliczana jest wartość dystrybuanty w zadanym punkcie położonym w pobliżu wyznaczonego kwantyla. Jeżeli wartość x będzie odległa od wyniku, to pojawi się komunikat ostrzegawczy, że wartości dystrybuanty nie można wyznaczyć.
Liczba wszystkich symulacji	long simulations_all()	qq.simulation_all()	Liczba wszystkich symulacji w kumulowanych obiektach kwantylowych.
Liczba użytych symulacji	long simulations_used()	qq.simulation_used()	Liczba symulacji w kumulowanych obiektach kwantylowych użytych do obliczania wyników (najczęściej równa liczbie wszystkich symulacji).

1.14 Język R (*rrun, rcppconv*)

Metoda	Deklaracja metody	Przykład wywołania	Opis działania

Domyślne środowisko R	RInside RE	RE	Aby wygodnie korzystać z R, w głównym pliku projektu (np. main.cpp) należy zadeklrować zmienną RE poza programem głównym. Zmienna RE będzie dostępna we wszystkich plikach projektu. Sposób deklaracji:
			
			Rinside RE;
			
			int main()
			{
			// kod programu
			}

Tekst	typedef Rcpp::String RText	RText t	Typ tekst w Rcpp. Zmienna tego typu może być przekazana do środowiska R.
Data	typedef Rcpp::Date RDate	RDate d	Typ data w Rcpp. Zmienna tego typu może być przekazana do środowiska R.
Macierz	typedef Rcpp::NumericMatrix RMatrix	RMatrix m	Typ macierz w Rcpp. Zmienna tego typu może być przekazana do środowiska R.
Wektor	typedef Rcpp::NumericVector RVector	RVector v	Typ wektor w Rcpp. Zmienna tego typu może być przekazana do środowiska R.
Tabela	typedef Rcpp::DataFrame RTable	RTable t	Typ tabela w Rcpp. Zmienna tego typu może być przekazana do środowiska R. Tabele w R nazywają się data.frame.
Typ dowolny	typedef RInside::Proxy	-	Typ dowolny. Funkcja R zwraca wynik typu RProxy, który jest następnie konkretyzowany. Patrz pomoc do funkcji „RProxy eval(Text)”.

	RProxy		
Konwertery obiektów			
Na tekst C++	Text ctext(const RText &)	ctext(tekst_R)	Konwertuję zmienną tekstową z R na zmienną tekstową C++.
Na tekst R	RText rtext(const Text &)	rtext(tekst_C)	Konwertuję zmienną tekstową z C++ na zmienną tekstową R.
Na datę C++	date cdate(const RDate &)	cdate(data_R)	Konwertuję zmienną data z R na zmienną data C++.
Na datę R	RDate rdate(const date &)	rdate(data_C)	Konwertuję zmienną data z C++ na zmienną data R.
Na macierz C++	matrix cmatrix(const RMatrix &)	cmatrix(macierz_R)	Konwertuję macierz z R na macierz C++.
Na macierz R	RMatrix rmatrix(const matrix &)	rmatrix(macierz_C)	Konwertuję macierz z C++ na macierz R.
Na wektor C++	matrix cvector(const RVector &)	cvector(wektor_R)	Konwertuję wektor z R na wektor C++.
Na wektor R	RVector rvector(const matrix &)	rvector(wektor_C)	Konwertuję wektor z C++ na wektor R.
Na tabelę C++	table ctable(const RTable &)	ctable(tabela_R)	Konwertuję tabelę z R (data.frame) na tabelę C++.
Na tabelę R	RTable rtable(const table &)	rtable(tabela_C)	Konwertuję tabelę z C++ na tabelę R (data.frame).
Przesyłanie zmiennych do środowiska R			
Przesyłanie zmiennych do R	void var(string,int) void var(string,double) void var(string,const Text &)	var("n",n) var("x",x) var("nazwa",nazwa) var("data1",data1)	Za pomocą funkcji var można przesłać zmienną z C++ do środowiska R. Nazwa zmiennej w R będzie taka jak pierwszy argument funkcji (string), a wartość taka jak drugi argument. W podanych przykładach nazwy zmiennych w R są takie same jak w C++, jednak nie jest to wymagane. To znaczy można napisać np.: var("x",y)

	void var(string,const date &) void var(string,const matrix &) void var(string,const table &)	var("A",A) var("tab",tab)	
Użycie zmienne w R	use(name)	use(x)	Zmienna C++ name staje się widoczna w R pod tą samą nazwą.
Deklarowanie zmiennej w R	let(name1,name2)	let(x,y)	W R powstaje zmienna name1, odpowiadająca zmiennej C++ name2.
Uruchamianie kodu R			
Uruchomienie kodu	void evalQ(Text)	evalQ("print(2)")	Funkcja uruchamia podany kod R. Działania są wykonywane w środowisku RE.
Uruchomienie kodu zwracającego wynik	RProxy eval(Text)	RVector wektorR=eval("rep(1, 10)")	Funkcja uruchamia podany kod R. Działania są wykonywane w środowisku RE. Wynik może być typu: int, double, RText, RDate, RMatrix, RVector lub RTable.
Uruchomienie kodu z pliku	void sourceQ(Text) void sourceQ(RInside &,Text)	sourceQ("funkcja.R") sourceQ(R1,"funkcja.R")	Funkcja uruchamia kod R z pliku o podanej nazwie. Domyślnie działania są wykonywane w środowisku RE, ale mogą być wykonywane w innych instancjach tego środowiska (jak w drugim przykładzie).
Uruchomienie kodu zwracającego wynik z pliku	RProxy source(Text) RProxy source(RInside &,Text)	RMatrix macierzR=source("funkcja2.R") RTable tabelaR=source(R1,"funkcja3.R")	Funkcja uruchamia kod R z pliku o podanej nazwie. Domyślnie działania są wykonywane w środowisku RE, ale mogą być wykonywane w innych instancjach tego środowiska (jak w drugim przykładzie). Wynik może być typu: int, double, RText, RDate, RMatrix, RVector lub RTable.

Makra uruchamiające	proc(name) fun(name) run(name)	proc(print(2)) fun(rep(1,10)) run(funkcja2)	Makra pozwalające uruchamiać kod z pominięciem cudzysłowów i rozszerzeń pliku (bardziej czytelne).