# Побудова найкращого чебишовського поліноміального наближення функцій однієї змінної

Header: *poly_apx.h*
Library: *libpoly_apx.a*

int PolyApproximation(int N, double x[], double y[], double& precision, int& na, double a[])

Виконує найкраще чебишовське наближення дискретно заданої функції алгебраїчним поліномом $\sum_{i=0}^{n} a_i x^i$ за заданою похибкою ε, яку не повинна перевищувати величина апостеріорної оцінки повної похибки поліноміального наближення δ. Алгоритм починає з наближення поліномом степеня $n$=0 і поступово підвищує його на одиницю, поки повна похибка наближення δ не задовольнить умову $\delta \leq \varepsilon$.

| Параметр | Тип | Опис |
|---|---|---|
| *N* | int | розмір масивів x і y |
| *x* | double * | масив значень аргументу |
| *y* | double * | масив значень функції, що наближується |
| *precision* | double & | ε |
| *na* | int & | кількість коефіцієнтів апроксимуючого поліному |
| *a* | double * | масив коефіцієнтів апроксимуючого поліному |

Функція повертає 0 при успішному завершенні та -1 у разі виникнення помилки.

class PolyFunc

Представляє алгебраїчний поліном. Чисто абстрактний клас.

*virtual double operator()(double x)*

Повертає значення поліному в точці x.

*virtual void GetCoeffs(std::valarray<double> &a) const*

Повертає масив коефіцієнтів поліному.

*virtual std::string GetStringRepresentation()*

Повертає поліном у вигляді рядка символів.

class PolyApproximator

Об'єкт цього класу виконує апроксимацію.

*PolyFunc *Approximate(const std::valarray<double> &Args, const std::valarray<double> &Func, double precision = 0)*

Параметри *Args* і *Func* відповідають параметрам *x* і *y* функції *PolyApproximation*, *precision* задає ε.

*double GetPrecision() const*

Функція повертає абсолютну похибку наближення.

Example

```
#include <iostream>
#include <valarray>
#include <string>
#include "poly_apx.h"


using namespace std;
using namespace icyb;


int main(int argc, char* argv[])
{
        double prec = 0.05;
        valarray<double> x(100), y(100);
        for ( int i = 0; i < 100; i ++ )
        {
                x[i] = 0.01*i;
                y[i] = exp(x[i]);
        }
```

```
        PolyApproximator PA;
        PolyFunc *FP = PA.Approximate(x, y, prec);

        cout << FP->GetStringRepresentation() << endl;
        cout << "precision: " << PA.GetPrecision() << endl;
        delete FP;
        return 0;
}
```

## Побудова найкращого чебишовського дробово-раціонального наближення функцій однієї змінної

Header: *rat_fraction.h*
Library: *librat_fraction.a*

int RFApproximation(int N, double x[], double y[], double& precision, int& na, int& nb, double a[], double b[])

Для підвищення точності наближення функцій, природа яких така, що функція має особливості ("сплески" - великі за модулем значення на деяких ділянках порівняно з іншими), доцільно застосовувати апроксимацію дробово-раціональними (нелінійними) виразами, які найбільш точно відображають особливості поведінки функції. Функція **RFApproximation** виконує найкраще чебишовське наближення дискретно заданої функції дробово-раціональним виразом $R_{mk}(x) = \frac{P_m(x)}{Q_k(x)} = \frac{\sum_{s=0}^{m} a_s x^s}{\sum_{t=0}^{k} b_t x^t}$ за заданою похибкою ε.

| Параметр | Тип | Опис |
|---|---|---|
| *N* | int | розмір масивів x і y |
| *x* | double * | масив значень аргументу |
| *y* | double * | масив значень функції, що наближується |
| *precision* | double & | ε |
| *na* | int & | кількість коефіцієнтів в чисельнику |
| *nb* | int & | кількість коефіцієнтів в знаменнику |
| *a* | double * | масив коефіцієнтів чисельнику |
| *b* | double * | масив коефіцієнтів знаменнику |

Функція повертає 0 при успішному завершенні та -1 у разі виникнення помилки.

class RFFunc

Представляє дробово-раціональний вираз $R_{mk}(x)$. Чисто абстрактний клас.

*virtual double operator()(double x)*

Повертає значення $R_{mk}(x)$ в точці *x*.

*virtual void GetCoeffs(std::valarray<double> &a, std::valarray<double> &b) const*

Записує коефіцієнти поліномів $P_m(x)$ і $Q_k(x)$ в масиви *a* і *b* відповідно.

*virtual std::string GetStringRepresentation()*

Повертає $R_{mk}(x)$ у вигляді рядка символів.

RFApproximator

Об'єкт цього класу виконує апроксимацію.

*RFFunc *Approximate(const std::valarray<double> &Args, const std::valarray<double> &Func, double precision = 0)*

Параметри *Args* і *Func* відповідають параметрам *x* і *y* функції *RFApproximation*, *precision* задає ε.

*double GetPrecision() const*

Функція повертає абсолютну похибку наближення.

Example

```
#include <iostream>
#include <valarray>
#include <string>
#include "rat_fraction.h"


using namespace std;
using namespace icyb;


int main(int argc, char* argv[])
{
        double prec = 0.05;
        valarray<double> x(100), y(100);
        for ( int i = 0; i < 100; i ++ )
        {
                x[i] = 0.01*i;
                y[i] = exp(x[i]);
        }


        RFApproximator RFA;
        RFFunc *FRF = RFA.Approximate(x, y, prec);


        cout << FRF->GetStringRepresentation() << endl;
        cout << "precision: " << RFA.GetPrecision() << endl;
        delete FRF;
        return 0;
}
```

## Побудова найкращого чебишовського наближення функцій багатьох змінних узагальненими поліномами

Header: *many_var.h*

Library: *libmany_var.a*

Будує найкраще рівномірне наближення функції $f(x)$ на множині *N* точок $\{x_1, x_2, \ldots, x_N\}$ узагальненим поліномом $F_n(x;Z)=\sum_{j=1}^{n} z_j \varphi_j(x)$, $Z=(z_1,\ldots,z_n)$, за системою *n* лінійно незалежних базисних функцій $\varphi_j(x)$.

class MVFunc

Представляє узагальнений поліном $F_n(x;Z)$. Чисто абстрактний клас.

*virtual double operator()(double x)*

Повертає значення $F_n(x;Z)$ в точці *x*.

*virtual void GetCoeffs(std::valarray<double> &a) const*

Повертає масив коефіцієнтів $z_j$ поліному $F_n(x;Z)$.

*virtual std::string GetStringRepresentation()*

Повертає поліном у вигляді рядка символів.

class MVApproximator

Об’єкт цього класу виконує апроксимацію.

*MVFunc *Approximate(const std::valarray<double> &Args, const std::valarray<double> &Func, double precision = 0, const std::string &Params = "")*

*Args* задає множину точок $\{x_1, x_2, \ldots, x_N\}$, *Func* – множину значень функції $\{f(x_1), f(x_2), \ldots, f(x_N)\}$. Система функцій $\varphi_j(x)$ задається параметром *Params* у вигляді рядка символів виду $"\varphi_1(x);\varphi_2(x);\ldots;\varphi_n(x)"$.

*double GetPrecision() const*

Функція повертає абсолютну похибку наближення.

Example

```
#include <iostream>
#include <valarray>
#include <string>
#include "many_var.h"

using namespace std;
using namespace icyb;

int main(int argc, char* argv[])
{
	valarray<double> x(100), y(100);
	for ( int i = 0; i < 100; i ++ )
	{
		x[i] = 0.01*i;
		y[i] = exp(x[i]);
	}

	MVApproximator SA;
	MVFunc *SF = SA.Approximate(x, y, 0.001, "1; x; x*x;");

	cout << SF->GetStringRepresentation() << endl;
	cout << "precision: " << SA.GetPrecision() << endl;
	delete SF;
	return 0;
}
```

## Наближення функцій багатьох змінних способом лінійної інтерполяції

Header: *lin_int.h*

Library: *many_var_interp.a*

int n_interpolation(int n, double ksi[], int m[], double x[], double y[], double &u, char *pBuf)

Функція **n_interpolation** інтерполює функцію багатьох змінних, що задана набором значень у дискретних точках, поліномами першої степені, обчислюючи значення у заданій точці ξ. Точки являють собою вузли сітки $E_N$ $n$-мірного простору, що задана значеннями $x_1^{(i)}, x_2^{(i)}, \ldots, x_{m_i}^{(i)}$ по кожній змінній: $E_N = \left\{x_1^{(1)}, x_2^{(1)}, \ldots, x_{m_1}^{(1)}\right\} \times \left\{x_1^{(2)}, x_2^{(2)}, \ldots, x_{m_2}^{(2)}\right\} \times \ldots \times \left\{x_1^{(n)}, x_2^{(n)}, \ldots, x_{m_n}^{(n)}\right\}$. Координати сітки задаються послідовністю $x_1^{(1)}, x_2^{(1)}, \ldots, x_{m_1}^{(1)}, x_1^{(2)}, x_2^{(2)}, \ldots, x_{m_2}^{(2)}, \ldots, x_1^{(n)}, x_2^{(n)}, \ldots, x_{m_n}^{(n)}$. Масив значень функції $f$ задається послідовністю

$$f\left(x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(n)}\right), f\left(x_2^{(1)}, x_1^{(2)}, \ldots, x_1^{(n)}\right), \ldots, f\left(x_{m_1}^{(1)}, x_1^{(2)}, \ldots, x_1^{(n)}\right),$$
$$f\left(x_1^{(1)}, x_2^{(2)}, x_1^{(3)}, \ldots, x_1^{(n)}\right), f\left(x_2^{(1)}, x_2^{(2)}, x_1^{(3)}, \ldots, x_1^{(n)}\right), \ldots, f\left(x_{m_1}^{(1)}, x_2^{(2)}, x_1^{(3)}, \ldots, x_1^{(n)}\right), \ldots,$$
$$f\left(x_{m_1}^{(1)}, x_{m_2}^{(2)}, x_1^{(3)}, \ldots, x_1^{(n)}\right), \ldots, f\left(x_{m_1}^{(1)}, x_{m_2}^{(2)}, x_{m_3}^{(3)}, \ldots, x_{m_{n-1}}^{(n-1)}, x_1^{(n)}\right), \ldots, f\left(x_{m_1}^{(1)}, x_{m_2}^{(2)}, x_{m_3}^{(3)}, \ldots, x_{m_n}^{(n)}\right).$$

| Параметр | Тип | Опис |
|---|---|---|
| *n* | int | Розмірність. |
| *ksi* | double * | Масив координат точки ξ розміру n. |
| *m* | int * | Масив розміру n кількості точок сітки по кожній координаті. m[i] ≥ 2, i = 0, n-1. |

| | | |
|---|---|---|
| *x* | double * | Масив координат сітки. |
| *y* | double * | Масив значень функції у вузлах сітки. |
| *u* | double & | Значення функції в точці ξ. |
| *pBuf* | char * | Масив байт розміру (3*n+1)*(sizeof(int)+sizeof(double)). |

Функція повертає 0 при успішному завершенні та -1, якщо точка ξ лежить поза межами сітки. *n*+1 коефіцієнт полінома типу double містить масив pBuf+(3*n+1)* sizeof(int)+2*n* sizeof(double).

class LinPol

Представляє функцію, яку буде отримано в результаті інтерполяції. Чисто абстрактний клас.

*virtual double operator()(const std::valarray<double> &x)*

Отримати значення функції в точці x.

*virtual void GetCoeffs(std::valarray<double> &a) const*

Отримати масив коефіцієнтів функції.

*virtual std::string GetStringRepresentation()*

Повертає функцію у вигляді рядка символів.

class LinApproximator

Об'єкт цього класу виконує інтерполяцію.

*LinPol *Approximate(const std::valarray<double> &ksi, const std::valarray<int> &m, const std::valarray<double> &x, const std::valarray<double> &y, double &u)*

Параметри функції аналогічні відповідним параметрам функції *n_interpolation*.

Example

```
#include <iostream>
#include <valarray>
#include <string>
using namespace std;
#include "lin_int.h"


int main(int argc, char *argv[])
{
	int n = 4;
	int m[] = {5, 2, 4, 3};
	int nx = 14, ny = 120;
	double x[] = {1.2, 2.0, 3.3, 5.8, 6.1, 0.1, 0.3, -2.5, -1.0, 0.8, 10.5, 13.2, 15.7, 23.3};
	double y[] = {2.5, 3.8, 2.2, 1.7, 5.2, 6.1, 7.9, 8.0, 7.8, 7.0, 6.3, 10.0, 12.5, 17.0,
21.0, 35.0, 34.1, 22.5, 17.3, 14.5, 12.1, 7.8, 0.0, -1.0, 1.0, -3.6, -7.8, -8.7, -12.9, -15.4,
-17.0, -30.0, -5.0, 4.0, 0.0, 1.2, 3.4, 5.6, 7.8, 9.1, 10.2, 11.3, 12.4, 15.5, 17.2, 9.3, 8.4,
11.5, 15.1, 13.2, 11.1, 9.9, 8.7, 7.8, 6.5, 5.6, 4.3, 2.0, 0.9, -1.0, -10.0, 5.0, 1.0, 4.0, 3.0,
2.0, 3.5, 10.1, 25.7, 31.4, 65.2, 30.5, 34.3, 36.0, 51.9, 40.3, 26.5, 10.9, 9.8, 11.7, 6.5, 4.0,
1.9, -5.7, -11.3, -21.1, -5.4, -3.6, 4.9, 7.1, 10.1, 12.3, 1.9, 4.5, 13.1, 15.3, 17.8, 12.6,
35.3, 37.9, 34.0, 29.9, 41.6, 80.0, 82.3, 81.4, 79.5, 60.7, 54.3, 49.1, 38.4, 34.5, 31.9, 27.4,
12.0, 9.1, 7.6, 5.0, 0.0, -1.2};
	double ksi[] = {1.2, 0.3, -1.5, 17.2};
	int i;


	char *pBuf = new char[(3*n+1)*sizeof(int)+(3*n+1)*sizeof(double)];
	double u;
	n_interpolation(n, ksi, m, x, y, u, pBuf);


	double *p = (double *)(pBuf+(3*n+1)*sizeof(int)+2*n*sizeof(double));
	cout << "coefficients:" << endl;
	for ( i = 0; i < n+1; i ++ )
		cout << p[i] << endl;
	cout << "\nu = " << u << endl;


	cout << "\n********************\n\n";
	std::valarray<double> va_ksi(ksi, n), va_x(x, nx), va_y(y, ny);
	std::valarray<int> va_m(m, n);
	icyb::LinApproximator LA;
	double w;
	icyb::LinPol *LP = LA.Approximate(va_ksi, va_m, va_x, va_y, w);
	if ( LP == NULL )
		std::cout << "error" << std::endl;
```

```
        else
                std::cout << LP->GetStringRepresentation() << "\nw = " << w << "\nw = " << (*LP)
(va_ksi) << std::endl;
        delete LP;


        delete [] pBuf;
        return 0;
}
```

## Інтерполяція функцій однієї змінної методом Ньютона

Header: *newton_int.h*
Library: *libnwt_int.a*

Бібліотека призначена для побудови інтерполяційного многочлена Ньютона для нерівних проміжків

$$L_n(x) = f(x_0) + (x - x_0) f(x_0; x_1) + \ldots + (x - x_0)(x - x_1)\ldots(x - x_{n-1}) f(x_0; x_1; \ldots; x_n),$$

$$f(x_i; x_{i+1}) = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i},$$

$$f(x_{i-1}; x_i; \ldots; x_{i+k}) = \frac{f(x_i; x_{i+1}; \ldots; x_{i+k}) - f(x_{i-1}; x_i; \ldots; x_{i+k-1})}{x_{i+k} - x_{i-1}}.$$

Ця форма запису інтерполяційного многочлена Лагранжа є більш зручною для обчислень, ніж формула Лагранжа, оскільки додавання нових вузлів інтерполяції не призводить до повторного обчислення обрахованих раніше розділених різниць.

void newton_int(const double *x, double *y, int n)

Обчислює масив розділених різниць.

| Параметр | Тип | Опис |
|---|---|---|
| *x* | const double * | Система вузлів інтерполяції $x_0, x_1, \ldots, x_n$. Масив може бути невідсортованим. |
| *y* | double * | На вході – масив значень функції *f* у точках $x_0, x_1, \ldots, x_n$. На виході y[0]= $f(x_0)$, y+1 – масив розділених різниць $f(x_0; x_1), \ldots, f(x_0; x_1; \ldots; x_n)$. |
| *n* | int | n+1 – розмір масивів x і y. |

double get_value(const double *x, const double *y, int n, double x0)

Обчислює значення інтерполяційного многочлена $L_n(x)$ в точці $x_0$ за схемою Горнера.

| Параметр | Тип | Опис |
|---|---|---|
| *x* | const double * | Система вузлів інтерполяції $x_0, x_1, \ldots, x_n$. |
| *y* | const double * | y[0]= $f(x_0)$, y+1 – масив розділених різниць $f(x_0; x_1), \ldots, f(x_0; x_1; \ldots; x_n)$. |
| *n* | int | n+1 – розмір масивів x і y. |
| *x0* | double | точка, в якій обчислюється значення $L_n(x)$ |

Функція повертає $L_n(x_0)$.

class CNwPol

Клас містить функції, що дозволяють побудувати і модифікувати інтерполяційний многочлен Ньютона $L_n(x)$.

*CNwPol(std::vector<double> &ax, std::vector<double> &ay)*

Будує інтерполяційний многочлен $L_n(x)$. Масив *ax* задає вузли інтерполяції, масив *ay* – значення функції у вузлах, на виході містить розділені різниці.

*double GetValue(double x) const*

Обчислює значення інтерполяційного многочлена в точці *x* за схемою Горнера.

*void AddNode(double x, double y)*

Додає новий вузол інтерполяції $x_{n+1} = x$ до існуючої системи вузлів $x_0, x_1, \ldots, x_n$ і будує інтерполяційний поліном $L_{n+1}(x)$ для функції $f$ по вузлам $x_0, x_1, \ldots, x_n, x_{n+1}$, $f(x_{n+1}) = y$.

Example

```
#include <vector>
#include "newton_int.h"


int main(int argc, char* argv[])
{
	double x[] = {0, 2, 3, 5}, y[] = {1, 3, 2, 5};
	std::vector<double> vx(4), vy(4);
	memcpy(&vx.front(), x, 4*sizeof(double));
	memcpy(&vy.front(), y, 4*sizeof(double));
	icyb::CNwPol nw_pol(vx, vy);
	printf("CNwPol::CNwPol():\n");
	for ( int i = 0; i < 4; i ++ )
		printf("vx[%d] = %g | vy[%d] = %g\n", i, vx[i], i, vy[i]);
	nw_pol.AddNode(6, 6);
	printf("CNwPol::AddNode():\n");
	for ( int i = 0; i < 5; i ++ )
		printf("vx[%d] = %g | vy[%d] = %g\n", i, vx[i], i, vy[i]);
	return 0;
}
```