САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

БУРОВА ИРИНА ГЕРАСИМОВНА

Практикум по методам вычислений
MAPLE, Open MP

Рецензенты:

Евдокимова Т.О., доцент Санкт-Петербургского гос.университета

Белякова О.В., доцент Калининградского гос.университета

Рекомендовано к опубликованию
УМК математико-механического факультета
19 января 2017 г. Протокол 1

Санкт-Петербург

2017

Бурова И Г

Практикум по методам вычислений.
MAPLE, OPEN MP

Содержание

1. ОРТОГОНАЛЬНЫЕ МНОГОЧЛЕНЫ

1.1. Многочлены Чебышева первого рода

Многочлены Чебышева первого рода определяются формулой

$$T_n(x) = \cos(n \arccos(x)), \quad |x| \le 1, \ n = 0, 1, 2, \ldots$$

Отсюда следует, что $T_0(x) = 1$, $T_1(x) = x$.

З а д а ч и

1. Построить последовательность многочленов Чебышева, используя рекуррентную формулу $T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x)$, $T_0(x) = 1, T_1(x) = x$, и приведенных многочленов Чебышева $\widetilde{T}_n(x) = T_n(x)/2^{n-1}$, $n = 2, 3, 4, 5, 6, 7$.

Указание. Для формирования последовательности многочленов можно использовать переменную T с индексом: $T[0] := 1; T[1] := x; T[2] := 2 * x^2 - 1;$

$for\ k\ from\ 2\ to\ n\ do\ T[k+1] := simplify(2*x*T[k]-T[k-1]);\ od;$

2. Построить графики многочленов Чебышева первого рода и приведенных многочленов первого рода на промежутке $[-1, 1]$ (команда *plot*).

3. Проверить с помощью аналитических вычислений ортогональность многочленов Чебышева на промежутке $[-1, 1]$ по весу $\frac{1}{\sqrt{1-x^2}}$:

$$I_{m,n} = \int_{-1}^{1} \frac{T_n(x)T_m(x)}{\sqrt{1-x^2}} dx = \begin{cases} 0, & m \ne n, \\ \pi/2, & m = n \ne 0, \\ \pi, & m = n = 0. \end{cases}$$

4. Найти полином наилучшего приближения $Q(x)$ в классе полиномов степени не выше $n-1$ для функции $f(x) = x^n$. Построить графики $Q(x)$ и $f(x) = x^n$ на промежутке $[-1, 1]$ в одних осях координат.

Указание. Вспомнить, что приведенный многочлен Чебышева $\widetilde{T}_n(x) = T_n(x)/2^{n-1}$ наименее уклоняется от нуля.

5. Доказать утверждения.

а) $T_n(-x) = (-1)^n T_n(x)$

б)

$$T_n(x) = det \begin{pmatrix} x & 1 & 0 & \dots & 0 & 0 \\ 1 & 2x & 1 & \dots & 0 & 0 \\ 0 & 1 & 2x & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2x & 1 \\ 0 & 0 & 0 & \dots & 1 & 2x \end{pmatrix},$$

где определитель имеет порядок n.

в) Доказать утверждение. Если $x^2 + y^2 = 1$, то

$$T_{2n}(y) = (-1)T_{2n}(x).$$

г) Доказать утверждение. Для произвольных целых m, n, справедливо тождество

$$T_m T_n = \frac{1}{2}(T_{m-n} + T_{m+n}).$$

1.2. Многочлены Чебышева второго рода

Многочлены Чебышева второго рода определяются формулой

$$U_n(x) = \sin((n+1)\arccos(x))/\sqrt{1-x^2}, \quad n = 0, 1, 2, \dots$$

Отсюда следует, что $U_0(x) = 1$, $U_1(x) = 2x$.

З а д а ч и

1. Построить последовательность многочленов Чебышева, используя рекуррентную формулу $U_{n+1}(x) = 2xU_n(x) - U_{n-1}(x)$, $n = 2, 3, 4, 5, 6, 7$.

2. Построить графики многочленов Чебышева второго рода (команда *plot*).

3. Проверить ортогональность многочленов Чебышева

$$I_{m,n} = \int_{-1}^{1} U_n(x)U_m(x)\sqrt{1-x^2}dx = \begin{cases} 0, & m \neq n, \\ \pi/2, & m = n. \end{cases}$$

4. Доказать утверждения.

а) $U_n(-x) = (-1)^n U_n(x)$,

б) для произвольного x верна формула

$$U_n(x) = \frac{1}{n+1} T'_{n+1}(x), \quad n \neq -1.$$

в)

$$U_n(x) = det \begin{pmatrix} 2x & 1 & 0 & \dots & 0 & 0 \\ 1 & 2x & 1 & \dots & 0 & 0 \\ 0 & 1 & 2x & \dots & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 2x & 1 \\ 0 & 0 & 0 & \dots & 1 & 2x \end{pmatrix}$$

где определитель имеет порядок n.

г) проверить равенства

$$T_n(x) = \frac{1}{2}(U_n(x) - U_{n-2}(x)),$$

$$T_n(x) = U_n(x) - xU_{n-1}(x),$$

$$T_n(x) = xU_{n-1}(x) - U_{n-2}(x),$$

$$U_n(x) = \frac{T_{n+2}(x) - T_n(x)}{2(x^2-1)},$$

$$U_n(x) = \frac{T_{n+2}(x) - xT_{n+1}(x)}{(x^2-1)},$$

$$U_n(x) = \frac{xT_{n+1}(x) - T_n(x)}{(x^2-1)},$$

д) Доказать утверждение. Для произвольных целых m, n, справедливо тождество

$$U_m T_n = \frac{1}{2}(U_{m-n} + U_{m+n}).$$

1.3. Многочлены Чебышева—Эрмита

Многочлены Чебышева—Эрмита определяются формулой

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}, \quad n \geq 0.$$

Отсюда следует, что $H_0 = 1$.

З а д а ч и

1. Построить последовательность многочленов Чебышева—Эрмита, используя рекуррентную формулу $H_{n+1}(x) = 2xH_n(x) - H_n'(x)$, $n = 2, 3, 4, 5, 6, 7$. Построить графики многочленов Чебышева—Эрмита (команда *plot*).

2. Проверить ортогональность многочленов Чебышева—Эрмита

$$I_{m,n} = \int_{-\infty}^{\infty} e^{-x^2} H_n(x) H_m(x) dx = \begin{cases} 0, & m \neq n, \\ 2^n \sqrt{\pi} n!, & m = n. \end{cases}$$

1.4. Многочлены Чебышева—Лагерра

Многочлены Чебышева—Лагерра определяются формулой

$$L_n^{\alpha}(x) = (-1)^n x^{-\alpha} e^x \frac{d^n}{dx^n}(x^{\alpha+n} e^{-x}), \quad \alpha > -1.$$

З а д а ч и

1. Построить последовательность многочленов Чебышева—Лагерра, $n = 2, 3, 4, 5, 6, 7$. Построить графики многочленов Чебышева—Лагерра.

2. Проверить ортогональность многочленов Чебышева—Лагерра

$$I_{m,n} = \int_0^{\infty} x^{\alpha} e^{-x} L_n^{(\alpha)}(x) L_m^{(\alpha)}(x) dx = \begin{cases} 0, & m \neq n, \\ n!\Gamma(n+\alpha+1), & m = n. \end{cases}$$

1.5. Многочлены Якоби

Многочлены Якоби определяются формулой

$$P_n^{\alpha,\beta}(x) = \frac{(-1)^n}{2^n n!}(1-x)^{-\alpha}(1+x)^{-\beta}\frac{d^n}{dx^n}[(1-x)^{\alpha+n}(1+x)^{\beta+n}],$$

$\alpha > -1, \beta > -1$.

З а д а ч и

1. Построить последовательность многочленов Якоби при $\alpha = \beta = 0$, $n = 3, 4, 5, 6, 7$. Построить графики многочленов Якоби при $\alpha = \beta = 0$.

2. Проверить ортогональность многочленов Якоби

$$I_{m,n} = \int_{-1}^{1}(1-x)^{\alpha}(1+x)^{\beta}P_n^{(\alpha,\beta)}(x)P_m^{(\alpha,\beta)}(x)dx = \begin{cases} 0, & m \neq n, \\ A = \text{const} \neq 0, & m = n. \end{cases}$$

1.6. Многочлены Лежандра

Многочлены Лежандра являются частным случаем многочленов Якоби при $\alpha = 0,\ \beta = 0$:

$$P_n(x) = \frac{1}{2^n n!}\frac{d^n}{dx^n}[(x^2-1)^n].$$

З а д а ч и

1. Построить последовательность многочленов Лежандра, $n = 2, 3, 4, 5, 6, 7$. Построить графики многочленов Лежандра.

2. Проверить ортогональность многочленов Лежандра

$$I_{m,n} = \int_{-1}^{1} P_n(x)P_m(x)dx = \begin{cases} 0, & m \neq n, \\ 2/(2n+1), & m = n. \end{cases}$$

2. ИНТЕРПОЛИРОВАНИЕ

2.1. Интерполяционный многочлен

З а д а ч а

Пусть на промежутке $[a, b]$, a, b — вещественные числа, задана сетка упорядоченных узлов X_k, $k = 0, 1, \ldots, n$, $f(X_k)$ — значения функции $f(x)$ в узлах сетки, далее предполагаем $f \in C^{(n+1)}[a, b]$. Требуется построить интерполяционный многочлен степени n для функции $f(x)$ в форме Лагранжа

$$P_n(x) = \sum_{k=0}^{n} f(X_k) \frac{w(x)}{(x - X_k)w'(X_k)},$$

где $w(x) = (x - X_0)(x - X_1) \ldots (x - X_n)$.

Как известно, остаток интерполяции имеет вид

$$f(x) - P_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w(x),\ \xi \in (a, b).$$

Указание. При разработке программы $f(X_k)$, X_k задать в виде массивов (в MAPLE элементы массивов нумеруются от единицы).

Возможный вариант программы. Производную от $w(x)$ по переменной x вычисляем и записываем в переменную $w1$:
$> \ w1 := diff(w, x) :$
Теперь интерполяционный многочлен может быть задан так:
$> \ P_n := sum(f[k] * w/(x - X[k])/(subs(x = X[k], w1)), k = 1..n + 1) :$
Затем упрощаем выражение:
$> \ P_n n := simplify(P_n) :$

Ниже приведен вариант решения задачи интерполяции функции $\sin(x)$ на равномерной сетке узлов с шагом $h = (b - a)/n$ и построения графика при $a = 0$, $b = 1$, $h = 0.1$ в MAPLE с применением функций *product*, *subs*:
$> restart;$
$> m := 10 : n := m + 1; d := 5; X := array(1..n) : \ fk := array(1..n) :$

```
> h := 1/m : for k from 1 by 1 to n do
    X[k] := (k-1)*h; fk[k] := sin(k*h); od :
> om1 := array(1..n) :
> o := product((x - X[k1]), k1 = 1..n);
> o1 := diff(o, x) :
> for k from 1 by 1 to n do om1[k] := subs(x = X[k], o1); od :
> pf := sum(fk[j]*o/(x - X[j])/om1[j], j = 1..n) :
> pf1 := normal(pf) :
> ss := array(1..d*n) :
> for k from 1 by 1 to d*n do ss[k] := subs(x = k*h/d, pf1); od :
> with(plots) :
> ss1 := array(1..d*n) : ss2 := array(1..d*n) :
> for k from 1 by 1 to d*n do
    ss1[k] := sin(k*h/d); ss2[k] := ss[k] - ss1[k]; od :
> pointplot(seq([ss[k1], ss2[k1]], k1 = 1..d*(n)));
```

Второй вариант решения на промежутке $[-1, 1]$, $h = 2/n$, с применением функции *unapply* и оформлением в виде процедуры:

```
> restart;
> n := 5 : h := 2/n; X := array(1..n+1) : f := array(1..n+1) :
> for k from 1 to n+1 do
            X[k] := -1 + h*(k-1); f[k] := sin(X[k]); od :
pr := proc(x)
w := unapply(product((x - X[k1]), k1 = 1..n+1), x);
w1 := unapply(simplify(diff(w(x), x)), x);
F := unapply(simplify(sum(f[k2]*w(x)/w1(X[k2])/(x - X[k2]),
            k2 = 1..n+1)), x);
RETURN(F(x)); endproc;
> z := (pr(t));
> m := n*10;
> for j from 1 to 2*m do
            t := -1 + j/m; rp[j] := evalf(z);
            rt[j] := sin(t); T[j] := t; od :
> ff := [seq([T[i2], rt[i2]], i2 = 1..2*m)] :
> ff1 := [seq([T[i2], rp[i2]], i2 = 1..2*m)] :
> with(plots) : ffa := pointplot(ff, color = red) :
> ffb := pointplot(ff1, color = green) : display(ffa, ffb);
```

Хотя, согласно теореме Веейрштрасса, любая непрерывная функция $f(x)$ на отрезке $[a,\ b]$ может быть как угодно хорошо приближена многочленами, ряд примеров показывает, что при достаточно большом числе узлов интерполяции нельзя гарантировать хорошее приближение интерполируемой функции.

С.Н.Бернштейн (1916 г.) рассмотрел для функции $f(x) = |x|$ на промежутке $[-1,\ 1]$ последовательность интерполяционных многочленов Лагранжа по равноотстоящим узлам при $x_0 = -1$, $x_n = 1$.

Рунге (1901 г.) показал, что интерполяционный процесс не сходится на промежутке $[-1,\ 1]$ для функции $f(x) = \frac{1}{1+25x^2}$.

В обоих случаях

$$\lim_{n\to\infty} \max_{-1\le x\le 1} |f(x) - P_n(x)| = +\infty.$$

З а д а ч и

1. Построить алгебраический интерполяционный многочлен по значениям $f(x_0),\ f(x_1),\ \ldots, f(x_n)$ на промежутке $[-1,\ 1]$ для следующих функций: а) $f = \mathrm{abs}(x)$; б) $f = 1\ (1+25x^2)$. Предполагаем, что узлы интерполяции равноотстоящие, $h = X_{k+1} - X_k = 2/n$. Результаты вычислений сравнить при $n = 3, 5, 7, 9, 11$.

Нарисовать в одних осях координат графики исходной функции и её приближения. Обратить внимание на ухудшение приближения вблизи концов промежутка с ростом n. Построить приближение этих функций на промежутке $[-1,\ 1]$, используя в качестве узлов интерполирования корни многочлена Чебышева 1 рода степени $n+1$. Сравните результаты вычислений, полученные на равномерной сетке узлов и на неравномерной (при использовании корней многочлена Чебышева 1 рода в качестве узлов интерполирования) при $n = 3, 5, 7, 9, 11$.

2. Построить функцию Лебега

$$L_n(x) = \sum_{k=0}^{n} \left| \frac{w(x)}{(x - X_k)w'(X_k)} \right|$$

при равномерной сетке узлов и при неравномерной (при использовании корней многочлена Чебышева 1 рода), $n = 3, 5, 7, 9, 11$. Объясните результат.

3. Найти оценку погрешности приближения при использовании корней многочлена Чебышева 1 рода в качестве узлов интерполяции.

На практике вместо интерполяционного многочлена высокой степени используют кусочно-многочленную функции (обычно на различных сеточных интервалах применяют многочлены одинаковой степени с разными коэффициентами так, чтобы в целом получилась функция с заданными свойствами). Такие кусочно-многочленные функции называют сплайнами. Простейший пример сплайна – ломаная (функция линейная на каждом сеточном интервале).

2.2. Интерполяционные эрмитовы кубические сплайны

Пусть в узлах сетки $a = x_0 < x_1 < \ldots < x_n = b$ заданы значения функции $f_i = f(x_i)$ и ее производной $f'_i = f'(x_i)$, $i = 0, 1, \ldots, n$. Построим кубический интерполяционный сплайн $S(x)$, так чтобы

$$S(x_i) = f_i, \quad S'(x_i) = f'_i,$$

т.е. $S \in C^1[a, b]$.

Для этого на каждом промежутке $[x_i, x_{i+1}]$, $i = 0, \ldots, n-1$, будем строить многочлен третьей степени

$$S_i(x) = a_{i0} + a_{i1}(x - x_i) + a_{i2}(x - x_i)^2 + a_{i3}(x - x_i)^3, \quad x \in [x_i, x_{i+1}], \quad (1)$$

так что

$$S_i(x_i) = f_i, \quad S'_i(x_i) = f'_i. \quad (2)$$

Из условий интерполяции (2) получаем систему уравнений

$$\begin{cases} S_i(x_i) = f_i, \\ S_(x_{i+1}) = f_{i+1}, \\ S'_i(x_i) = f'_i, \\ S'_i(x_{i+1}) = f'_{i+1}. \end{cases}$$

Решение этой системы уравнений можно записать в виде (см.[4])

$$S_i(x) = \varphi_1(t)f_i + \varphi_2(t)f_{i+1} + \varphi_3(t)h_i f'_i + \varphi_4(t)h_i f'_{i+1}, \qquad (3)$$

где $\varphi_1(t) = (1-t)^2(1+2t)$, $\varphi_2(t) = t^2(3-2t)$, $\varphi_3(t) = t(1-t)^2$, $\varphi_4(t) = -t^2(1-t)$. $h_i = x_{i+1} - x_i$, $t = (x - x_i)/h_i$. Формула (3) удобна для теоретических вычислений.

Для практических вычислений преобразуем (3) и будем использовать расчетную формулу

$$S_i(x) = f_i + (x - x_i)[f'_i + t(B + tA)], \quad x \in [x_i, x_{i+1}),$$

где $A = -2(f_{i+1} - f_i)/h_i + (f'_i + f'_{i+1})$, $B = -A + (f_{i+1} - f_i)/h_i - f'_i$. $h_i = x_{i+1} - x_i$, $t = (x - x_i)/h_i$.

З а д а ч а

Построить интерполяционный кубический эрмитов сплайн по значениям $f(x_0)$, $f(x_1)$, $\ldots, f(x_n)$, $f'(x_0)$, $f'(x_1)$, $\ldots, f'(x_n)$ функции $f(x) = 1/(1+25x^2)$ в узлах равномерной сетки узлов $\{x_i\}$ с шагом h: $x_i = h$, $i = 1, \ldots, n$, $n = 5, 10, 20$. Нарисовать графики функции и приближения в одних осях координат.

Возможное решение:

```
> restart;
> f := unapply(1/(1+25*x^2), x):
> f1 := unapply(diff(f(x), x), x):
> n := 4; h := 1/n:
> for i from 1 to 2*n+1 do
    X[i] := -1+i*h-h; F[i] := f(X[i]); F1[i] := f1(X[i]); od:
> d := 10; h1 := h/d;
> for i from 1 by 1 while i < 2*n+1 do
    for j from 0 to 9 do t := (j*h1)/h;
    A := -2*(F[i+1]-F[i])/h+F1[i]+F1[i+1];
    B := -A+(F[i+1]-F[i])/h-F1[i]; xx[10*i+j] := X[i]+j*h1;
    s[10*i+j] := F[i]+j*h1*(F1[i]+t*(B+t*A)); od; od;
> with(plots): ff := [seq([xx[i2], s[i2]], i2 = d..d*(2*n)+9)]:
> ff1 := pointplot(ff): display(ff1);
```

2.3. Кубические B-сплайны

Пусть в узлах сетки $a = x_0 < x_1 < \ldots < x_n = b$ заданы значения функции $f_j = f(x_j)$, $j = 0, 1, \ldots, n$. На промежутке $[x_j, x_{j+1}]$ будем строить кубический интерполяционный сплайн

$$S(x) = a_{j0} + a_{j1}(x - x_j) + a_{j2}(x - x_j)^2 + a_{j3}(x - x_j)^3, \qquad (4)$$

так что

$$S(x_j) = f_j, \qquad (5)$$

$S \in C^2[a, b]$. Поэтому $S^{(r)}(x_j-) = S^{(r)}(x_j+)$, $j = 1, \ldots, n-1$, $r = 0, 1, 2$. Всего неизвестных коэффициентов $4n$, условий $3(n-1)+n+1 = 4n-2$.

Дополнительно задается одно из следующих граничных условий:

1. $S'(a) = f'(a), \quad S'(b) = f'(b)$.
2. $S''(a) = f''(a), \quad S''(b) = f''(b)$.
3. $S^{(r)}(a) = S^{(r)}(b)$, $r = 1, 2$.
4. $S'''(x_j+) = S'''(x_j-)$.

1. Пусть дано граничное условие $S'(a) = f'(a), \quad S'(b) = f'(b)$. Введем обозначения $S'(x_i) = m_i$, $i = 0, \ldots, n$.

$$S(x) = \varphi_1(t) f_i + \varphi_2(t) f_{i+1} + \varphi_3(t) h m_i + \varphi_4(t) h m_{i+1}, \qquad (6)$$

где

$$\varphi_1(t) = (1-t)^2(1+2t), \quad \varphi_2(t) = t^2(3-2t),$$
$$\varphi_3(t) = t(1-t)^2, \quad \varphi_4(t) = -t^2(1-t),$$
$$h = x_{i+1} - x_i, \quad t = \frac{(x - x_i)}{h}.$$

На $[x_i, x_{i+1}]$ имеем

$$S''(x) = (f_{i+1} - f_i)\frac{(6-12t)}{h^2} + m_i\frac{(6t-4)}{h} + m_{i+1}\frac{(6t-2)}{h}.$$

Поэтому

$$S''(x_i+) = 6\frac{(f_{i+1}-f_i)}{h^2} - m_i\frac{4}{h} + m_{i+1}\frac{(-2)}{h}.$$

На $[x_{i-1}, x_i]$ имеем

$$S''(x) = (f_i - f_{i-1})\frac{(6-12t)}{h^2} + m_{i-1}\frac{(6t-4)}{h} + m_i\frac{(6t-2)}{h}.$$

Поэтому

$$S''(x_i-) = -6\frac{(f_i-f_{i-1})}{h^2} - m_{i-1}\frac{2}{h} + m_i\frac{4}{h}.$$

Условие

$$S''(x_i+) = S''(x_i-)$$

дает

$$6\frac{(f_{i+1}-f_i)}{h^2} - m_i\frac{4}{h} + m_{i+1}\frac{(-2)}{h} =$$
$$= -6\frac{(f_i-f_{i-1})}{h^2} - m_{i-1}\frac{2}{h} + m_i\frac{4}{h}.$$

Обозначим $\mu_i = 1/2$, тогда после преобразований с учетом граничных условий получим систему уравнений

$$\begin{cases} \mu_i m_{i+1} + 2m_i + \mu_i m_{i-1} = c_i, \quad i = 1, \ldots, n-1, \\ m_0 = f'(a), \\ m_N = f'(b), \end{cases}$$

где

$$c_i = 3(f_i - f_{i-1})\frac{\mu_i}{h} + 3\frac{(f_{i+1}-f_i)\mu_i}{h}.$$

Приведем теоремы о кубическом B-сплайне (см.[5]).

Теорема 1. Интерполяционный кубический B-сплайн, удовлетворяющий одному из условий 1–4, существует и единствен.

Д о к а з а т е л ь с т в о. Матрица системы уравнений имеет диагональное преобладание, и, таким образом, система имеет единственное решение.

Введем обозначения:

$$\omega_i(f) = \max_{x',x''\in[x_i,x_{i+1}]} |f(x') - f(x'')|,$$

$$\omega(f) = \max_{0\le i\le n-1} \omega_i(f).$$

Теорема 2. Если $f \in C[a,b]$ удовлетворяет условиям 1 или 3, то

$$\|S(x) - f(x)\|_C \le \Big(1 + \frac{3}{4}\Big)\omega(f).$$

Д о к а з а т е л ь с т в о. Пусть $x \in [x_i, x_{i+1}]$. Имеем

$$|S(x) - f(x)| \le ht(1-t)[(1-t)|m_i| + t|m_{i+1}|] +$$
$$+|f_i(1-t)^2(1+2t) + f_{i+1}t^2(3-2t) - f(x)|.$$

По теореме о среднем

$$f_i(1-t)^2(1+2t) + f_{i+1}t^2(3-2t) = f(\xi).$$

Таким образом,

$$|S(x) - f(x)| \le \omega(f) + \frac{1}{4}h\max\{|m_i|, |m_{i+1}|\}.$$

Итак, при построении кубических B-сплайнов нужно решить систему уравнений

$$\mu_i m_{i+1} + 2m_i + \mu_i m_{i-1} = c_i, \quad i = 1, \dots, n-1,$$
$$m_0 = f'(a),$$
$$m_n = f'(b),$$

где $h_i = x_{i+1} - x_i$, $\mu_i = 1/2$,

$$c_i = 3(f_i - f_{i-1})\frac{\mu_i}{h} + 3\frac{(f_{i+1} - f_i)\mu_i}{h}.$$

Решая эту систему уравнений находим m_i и далее берем $f_i' = m_i$, в формулах для эрмитового кубического сплайна при $x \in [x_i, x_{i+1}]$,

$$S_i(x) = f_i + (x - x_i)[f_i' + t(B + tA)],$$

где

$$A = -2\frac{f_{i+1} - f_i}{h} + (f'_i + f'_{i+1}),$$

$$B = -A + \frac{f_{i+1} - f_i}{h} - f'_i, \quad t = \frac{(x - x_i)}{h}.$$

З а д а ч а

1. Построить кубический B-сплайн по значениям функции $f(x) = 1/(1 + 25x^2)$ в узлах равномерной сетки узлов $\{x_i\} \in [-1, 1]$ с шагом h: $x_i = -1 + jh$, $j = 0, \ldots, n$, $n = 5$ (для решения системы уравнений использовать команды *solve* или *linsolve*). Построить график функции и ее приближения кубическим B-сплайном при $n = 10, 50$.

Далее будут рассмотрены вопросы приближения минимальными интерполяционными сплайнами.

3. О СПЛАЙНОВЫХ АППРОКСИМАЦИЯХ МАКСИМАЛЬНОГО ДЕФЕКТА

3.1. О полиномиальных лагранжевых сплайновых аппроксимациях

Пусть функция $u \in C^{m+1}(R^1)$, $\{x_j\}$ – сетка упорядоченных узлов

$$\ldots < x_{j-1} < x_j < x_{j+1} \ldots$$

такая, что для некоторого $K_0 > 1$ выполняется соотношение

$$K_0^{-1} \leq \frac{x_{j+1} - x_j}{x_j - x_{j-1}} \leq K_0.$$

Нетрудно видеть, что для равномерной сетки с шагом h $K_0 = 1$. Аппроксимацию $\widetilde{u}(x)$ функции u на промежутке $[x_k, x_{k+1})$ будем строить в виде

$$\widetilde{u}(x) = \sum_j u(x_j)\omega_j(x), \tag{1}$$

где $\omega_j(x)$ – интерполяционный базис полиномиальных сплайнов (см. [1]), определяемый из условия точности аппроксимации (1) на полиномах степени m.

$$\widetilde{u}(x) = u(x),\ u = 1, x, \ldots, x^m.$$

При построении сплайнов $\omega_j(x)$ предполагаем, что

$$\text{supp } \omega_j(x) = [x_{j-r-\alpha}, x_{j+r_1-\alpha}],$$

где r и r_1 – некоторые натуральные числа, задающие носитель сплайна $\omega_j(x)$, а α принимает одно из значений $\alpha = -r+1, \ldots, r_1-1$. Будем предполагать, что $m = r + r_1 - 1$. Считая, что кратность накрытия произвольной точки x носителями базисных сплайнов $\omega_j(x)$ ограничена числом $m+1$, нетрудно заметить, что в сумме выражения (1) небольшое количество слагаемых:

$$\widetilde{u}(x) = \sum_{j=k-r_1+1-\alpha}^{k+r-\alpha} u(x_j)\omega_j(x). \tag{2}$$

При $r + r_1 = m + 1$ сплайны $\omega_j(x)$ определяются однозначно.

Система уравнений

$$\sum_{j=k-r_1+1-\alpha}^{k+r-\alpha} \omega_j(x) x_j^\nu = x^\nu, \quad \nu = 0, \ldots, m$$

при $x \in [x_k, x_{k+1}]$ однозначно задает минимальные интерполяционные сплайны $\omega_j(x)$:

$$\omega_j(x) = \begin{cases} \prod\limits_{\substack{j' \neq j \\ -r_1+1 \le j'-k \le r}} \dfrac{(x - x_{j'})}{(x_j - x_{j'})}, & x \in [x_{k-\alpha}, x_{k+1-\alpha}), \\ & k = j - r, \ldots, j + r_1 - 1, \\ 0, & x \notin [x_{j-r-\alpha},\ x_{j+r_1-\alpha}]. \end{cases} \tag{3}$$

Заметим, что приближенные значения производных функции u легко вычисляются, если известны производные $\omega_j^{(\beta)}(x)$:

$$\widetilde{u}^{(\beta)}(x) = \sum_{j=k-r_1+1-\alpha}^{k+r-\alpha} u(x_j) \omega_j^{(\beta)}(x).$$

Построенные таким образом сплайны $\omega_j(x)$ имеют следующие свойства:

1) $\omega_j(x_j) = 1$, $\omega_j(x_k) = 0$ при $j \neq k$,

2) supp $\omega_j = [x_{j-r-\alpha}, x_{j+r_1-\alpha}]$,

3) $\omega_j \in C(R^1)$.

5) $|u - \widetilde{u}| \le h^{m+1} K \|u^m\|$, $|u^{(\beta)} - \widetilde{u}^{(\beta)}| \le h^{m+1-\beta} K \|u^m\|$, где $\beta = 1, \ldots, m$, $h = \max\limits_j |x_{j+1} - x_j|$, $K = \text{const}$ – зависит от ω_j и K_0 и не зависит от u.

В соответствии с выбором вершины j базисного сплайна $\omega_j(x)$ относительно носителя базисные сплайны (3) будем называть левыми, если $r + \alpha \le [\frac{m+1}{2}]$, и, соответственно, правыми, если $r_1 - \alpha \le [\frac{m+1}{2}]$. Здесь и далее $[m]$ – наибольшая целая часть числа m, не превосходящая это число.

Отметим, что вблизи левого конца рекомендуется использовать правые базисные сплайны, а вблизи правого конца – левые базисные сплайны.

Далее будем рассматривать конечную сетку

$$a = x_0 < x_1 < \ldots < x_N = b.$$

Пусть $m_1 = [\frac{m}{2}]$. Выбор базисных функций (3) для аппроксимации (2) можно осуществить, например, следующим образом. На промежутках $[x_i, x_{i+1}]$, $i = 0, \ldots, m_1$ берем $r_1 = 1$, $\alpha = -i$. На промежутках $[x_k, x_{k+1}]$ при $k = m_1 + 1, \ldots, N - m_1 - 1$ полагаем $r_1 = 1$, $\alpha = -m_1$, На промежутках $[x_{N-m_1+i}, x_{N-m_1+i+1}]$, $i = 0, \ldots, m_1 - 1$ задаем $r = 1$, $\alpha = m_1 - i - 1$.

3.1.1. Частные случаи

а) При $j = 0, \ldots, N-1$ функцию u приближаем выражением

$$\widetilde{u}(x) = u(x_j)\omega_j(x) + u(x_{j+1})\omega_{j+1}(x), \tag{4}$$

где

$$\omega_j = \begin{cases} \dfrac{x - x_{j+1}}{x_j - x_{j+1}}, & x \in [x_j, x_{j+1}), \\ \dfrac{x - x_{j-1}}{x_j - x_{j-1}}, & x \in [x_{j-1}, x_j), \\ 0, & x \notin [x_{j-1}, x_{j+1}]. \end{cases} \tag{5}$$

б) Пусть $\alpha = 0$, $r_1 = 2$, $r = 1$. Аппроксимация $\widetilde{u}$ на промежутке $[x_0, x_1]$ задается выражением

$$\widetilde{u}(x) = u(x_0)\omega_0(x) + u(x_1)\omega_1(x) + u(x_2)\omega_2(x), \tag{6}$$

где гранично-минимальные сплайны $\omega_j(x)$ на промежутке $[x_0, x_1]$ определяются формулами

$$\begin{cases} \omega_0(x) = \dfrac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}, \\ \omega_1(x) = \dfrac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}, \\ \omega_2(x) = \dfrac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}. \end{cases} \tag{7}$$

На промежутках $[x_k, x_{k+1}]$, $k = 1, \ldots, N-1$ используем приближение

$$\widetilde{u}(x) = u(x_{j-1})\omega_{j-1}(x) + u(x_j)\omega_j(x) + u(x_{j+1})\omega_{j+1}(x), \qquad (8)$$

где $\omega_j(x)$, $j = 1, \ldots, N$ задаются формулами

$$\omega_j(x) = \begin{cases} \dfrac{(x - x_{j-1})(x - x_{j-2})}{(x_j - x_{j-1})(x_j - x_{j-2})}, & x \in [x_{j-1}, x_j), \\ \dfrac{(x - x_{j-1})(x - x_{j+1})}{(x_j - x_{j-1})(x_j - x_{j+1})}, & x \in [x_j, x_{j+1}), \\ \dfrac{(x - x_{j+1})(x - x_{j+2})}{(x_j - x_{j+1})(x_j - x_{j+2})}, & x \in [x_{j+1}, x_{j+2}), \\ 0, & x \notin [x_{j-1}, x_{j+2}]. \end{cases} \qquad (9)$$

в) Пусть $\alpha = 0$, $r_1 = 1$, $r = 2$. На промежутках $[x_k, x_{k+1}]$, $k = 0, \ldots, N-2$ используем приближение

$$\widetilde{u}(x) = u(x_j)\omega_j(x) + u(x_{j+1})\omega_{j+1}(x) + u(x_{j+2})\omega_{j+2}(x), \qquad (1.10)$$

где гранично-минимальные сплайны определяются выражениями

$$\omega_j(x) = \begin{cases} \dfrac{(x - x_{j-2})(x - x_{j-1})}{(x_j - x_{j-2})(x_j - x_{j-1})}, & x \in [x_{j-2}, x_{j-1}), \\ \dfrac{(x - x_{j-1})(x - x_{j+1})}{(x_j - x_{j-1})(x_j - x_{j+1})}, & x \in [x_{j-1}, x_j), \\ \dfrac{(x - x_{j+1})(x - x_{j+2})}{(x_j - x_{j+1})(x_j - x_{j+2})}, & x \in [x_j, x_{j+1}), \\ 0 & x \notin [x_{j-2} - x_{j+1}], \end{cases} \qquad (1.11)$$

а на промежутке $[x_{N-1}, x_N]$ функцию u аппроксимируем выражением

$$\widetilde{u}(x) = u(x_{N-2})\omega_{N-2}(x) + u(x_{N-1})\omega_{N-1}(x) + u(x_N)\omega_N(x), \qquad (12)$$

где сплайны $\omega_j(x)$ определяются формулами

$$\begin{cases} \omega_{N-2}(x) = \dfrac{(x - x_N)(x - x_{N-1})}{(x_{N-2} - x_N)(x_{N-2} - x_{N-1})}, \\ \omega_{N-1}(x) = \dfrac{(x - x_N)(x - x_{N-2})}{(x_{N-1} - x_N)(x_{N-1} - x_{N-2})}, \\ \omega_N(x) = \dfrac{(x - x_{N-1})(x - x_{N-2})}{(x_N - x_{N-1})(x_N - x_{N-2})}. \end{cases} \qquad (13)$$

З а д а ч и

1. Построить интерполяционный кусочно-линейный и кусочно-квадратичный полиномиальный сплайн по значениям $f(x_0)$, $f(x_1)$, $\ldots, f(x_n)$. Получить оценку погрешности приближения функции f кусочно-линейным сплайном в норме $\|f\|_{C[a,b]} = \max\limits_{[a,b]} |f|$, если $f \in C^1[a,b]$, $f \in C^2[a,b]$.

Получить оценку погрешности приближения функции f кусочно-квадратичным сплайном если $f \in C^3[a,b]$.

Выписать основные характеристики (степень, гладкость, порядок аппроксимации) построенных приближений.

2. Пусть дана таблица натуральных логарифмов чисел от 1000 до 10000. Какова наибольшая погрешность линейной интерполяции, если шаг равен 1?

3. Решить задачу 2 для $f(x) = \sqrt(x)$ в промежутке $[0, 1.5]$.

4. Решить задачу 2 для $f(x) = \frac{1}{\pi}\int_0^x e^(- z^2)dz$ при $x \in [0,2]$, $h = 0.001$.

5. Найти выражения для базисных сплайнов, обладающих свойством точности на: а) $\varphi_k(x) = \sin(kx)$, $k = 1,2,3$; б) $\varphi_k(x) = \cos(kx)$, $k = 0,1,2$.

6. Рассмотрим сетку узлов, удовлетворяющую неравенствам $K_0^{-1} \le (x_{j+1} - x_j)/(x_j - x_{j-1}) \le K_0$, где $K_0 > 0$. Какой вид приобретает оценка погрешности? *Указание.*

7. Рассмотрим на плоскости прямоугольную область G_2 Выберем параметры $0 < h < 1$ и $0 < \tau < 1$. На осях x и y построим сетки узлов $\{x_j\}$ и $\{y_j\}$. Проведем два семейства параллельных прямых

$$x = x_0 + jh, j = 0, \pm 1, \pm 2, \ldots,$$

$$y = y_0 + k\tau, k = 0, \pm 1, \pm 2, \ldots$$

Точки пересечения этих прямых — узлы сетки в прямоугольнике. Два узла будем считать соседними, если они удалены друг от друга в направлении оси x или оси y на расстояние шага сетки вдоль этой оси. Будем рассматривать только узлы, принадлежащие $\overline{G}$. Те из них, у которых все четыре соседних узла принадлежат этому

множеству, назовем внутренними. Узлы, у которых хоть один соседний не принадлежит G^* – граничными. Множество внутренних узлов обозначим G^*.

Наряду с равномерной сеткой по осям x и y будем рассматривать сетку узлов, удовлетворяющую неравенствам

$$K_0^{-1} \le \frac{x_{j+1} - x_j}{x_j - x_{j-1}} \le K_0, \quad K_1^{-1} \le \frac{y_{k+1} - y_k}{y_k - y_{k-1}} \le K_1,$$

где $K_0 > 0$, $K_1 > 0$.

Пусть базисный сплайн $\omega_j(x)$ имеет носитель $[x_{j-s_1}, x_{j+l_1}]$, а базисный сплайн $\omega_k(y)$ имеет носитель $[x_{k-s_2}, x_{k+l_2}]$. Построим базисный сплайн $\Omega_{jk}(x, y)$ по правилу $\Omega_{jk}(x, y) = \omega_j(x)\omega_k(y)$. Будем называть $\Omega_{jk}(x, y)$ мультипликативными базисными функциями. Очевидно, что носитель Ω_{jk} есть прямоугольник с вершинами (x_{j-s_1}, y_{k-s_2}), (x_{j+l_1}, y_{k-s_2}), (x_{j-s_1}, y_{k+l_2}), (x_{j+l_1}, y_{k+l_2}).

Убедитесь, что $\Omega_{jk}(x_j, y_k) = 1$ и $\Omega_{jk}(x_{j'}, y_{k'}) = 0$ при $j \ne j'$ и $k \ne k'$.

Пусть $u(x_j, y_k)$ — значение функции $u(x, y)$ в узле (x_j, y_k). Приближение $\widetilde{U}(x, y)$ в прямоугольнике с вершинами (x_j, y_k), (x_{j+1}, y_k), (x_j, y_{k+1}), (x_{j+1}, y_{k+1}) будем строить по формуле

$$\widetilde{U}(x, y) = \sum_j \sum_k u(x_j, y_k)\Omega_{jk}(x, y).$$

Выписать оценку погрешности приближения.

3.2. О тригонометрических сплайновых аппроксимациях лагранжевого типа

Пусть функция $u \in C^{2m+1}(R^1)$, $\{x_j\}$ – сетка упорядоченных узлов

$$\ldots < x_{j-1} < x_j < x_{j+1} \ldots$$

такая, что для некоторого $K_0 > 1$ выполняется соотношение

$$K_0^{-1} \le \frac{x_{j+1} - x_j}{x_j - x_{j-1}} \le K_0.$$

Нетрудно видеть, что для равномерной сетки с шагом h $K_0 = 1$. Аппроксимацию $\widetilde{u}(x)$ функции u на промежутке $[x_k, x_{k+1})$ будем строить в виде

$$\widetilde{u}(x) = \sum_j u(x_j)\omega_j(x), \tag{1}$$

где $\omega_j(x)$ – интерполяционный базис тригонометрических сплайнов, определяемый из условия точности аппроксимации (1) на тригонометрических многочленах порядка m.

$$\widetilde{u}(x) = u(x),\ u = 1, \sin(x), \cos(x) \ldots, \sin(mx), \cos(mx).$$

При построении сплайнов $\omega_j(x)$ предполагаем, что

$$\text{supp } \omega_j(x) = [x_{j-r}, x_{j+r_1}],$$

где r и r_1 – некоторые натуральные числа, задающие носитель сплайна $\omega_j(x)$. Будем предполагать, что $2m = r + r_1 - 1$. Считая, что кратность накрытия произвольной точки x носителями базисных сплайнов $\omega_j(x)$ ограничена числом $2m+1$, нетрудно заметить, что в сумме выражения (1) небольшое количество слагаемых:

$$\widetilde{u}(x) = \sum_{j=k-r_1+1}^{k+r} u(x_j)\omega_j(x). \tag{2}$$

При $r + r_1 = 2m + 1$ сплайны $\omega_j(x)$ определяются однозначно.

Система уравнений

$$\sum_{j=k-r_1+1}^{k+r} \omega_j(x)\sin(\nu x_j) = \sin(\nu x),$$

$$\sum_{j=k-r_1+1}^{k+r} \omega_j(x)\cos(\nu x_j) = \cos(\nu x),$$

$\nu = 0, \ldots, m$ при $x \in [x_k, x_{k+1}]$ однозначно задает минимальные интерполяционные сплайны $\omega_j(x)$:

$$\omega_j(x) = \begin{cases} \prod\limits_{\substack{j' \neq j \\ -r_1+1 \leq j'-k \leq r}} \dfrac{\sin(x/2 - x_{j'}/2)}{sin(x_j/2 - x_{j'}/2)}, & x \in [x_k, x_{k+1}), \\ & k = j-r, \ldots, j+r_1-1, \\ 0, & x \notin [x_{j-r},\ x_{j+r_1}]. \end{cases} \tag{3}$$

Заметим, что приближенные значения производных функции u легко вычисляются, если известны производные $\omega_j^{(\beta)}(x)$:

$$\widetilde{u}^{(\beta)}(x) = \sum_{j=k-r_1+1}^{k+r} u(x_j)\omega_j^{(\beta)}(x).$$

4. РЕШЕНИЕ СИСТЕМ ЛИНЕЙНЫХ УРАВНЕНИЙ

4.1. Метод прогонки

При решении некоторых задач (например, построения кубического B-сплайна) приходится решать систему линейных алгебраических уравнений с матрицей ленточного вида. Рассмотрим систему линейных алгебраических уравнений с трехдиагональной матрицей

$$a_k y_{k-1} + b_k y_k + c_k y_{k+1} = f_k, \quad k = 1, \ldots, n, \tag{1}$$

где $a_1 = c_1 = 0$, $|b_k| > |a_k| + |c_k|$, $k = 1, \ldots, n$. Рассмотрим последовательный и параллельный алгоритм решения системы уравнений методом прогонки.

Для наглядности представления алгоритма возьмем $n = 6$. Пусть A — расширенная матрица системы.

$$A = \begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 & f_1 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 & f_2 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 & f_3 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 & f_4 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 & f_5 \\ 0 & 0 & 0 & 0 & a_6 & b_6 & f_6 \end{pmatrix}$$

Последовательный алгоритм

Первое уравнение $b_1 y_1 + c_1 y_2 = f_1$ перепишем в виде

$$y_1 = K_2 y_2 + L_2, \tag{2}$$

где $K_2 = -\frac{c_1}{b_1}$, $L_2 = \frac{f_1}{b_1}$. Из второго уравнения системы

$$a_2 y_1 + b_2 y_2 + c_2 y_3 = f_2, \tag{3}$$

после подстановки в него (2),

$$a_2(K_2 y_2 + L_2) + b_2 y_2 + c_2 y_3 = f_2,$$

получим

$$y_2 = K_3 y_3 + L_3,$$

где

$$K_3 = \frac{-c_2}{a_2K_2 + b_2}, \quad L_3 = \frac{-a_2L_2 + f_2}{a_2K_2 + b_2}.$$

Аналогично выводим

$$y_{j-1} = K_j y_j + L_j, \quad j = 4, \ldots, 6, \tag{4}$$

где

$$K_j = \frac{-c_{j-1}}{a_{j-1}K_{j-1} + b_{j-1}}, \quad L_j = \frac{-a_{j-1}L_{j-1} + f_{j-1}}{a_{j-1}K_{j-1} + b_{j-1}}.$$

Далее

$$y_5 = K_6 y_6 + L_6, \tag{5}$$

подставляем в последнее уравнение

$$a_6 y_5 + b_6 y_6 = f_6,$$

Отсюда получаем

$$y_6 = \frac{-a_6L_6 + f_6}{a_6K_6 + b_6},$$

Поэтому в общем случае

$$A = \begin{pmatrix} b_1 & c_1 & 0 & 0 & 0 & 0 & f_1 \\ a_2 & b_2 & c_2 & 0 & 0 & 0 & f_2 \\ 0 & a_3 & b_3 & c_3 & 0 & 0 & f_3 \\ 0 & 0 & a_4 & b_4 & c_4 & 0 & f_4 \\ 0 & 0 & 0 & a_5 & b_5 & c_5 & f_5 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & a_n & b_n & f_n \end{pmatrix}$$

$$y_n = \frac{-a_nL_n + f_n}{a_nK_n + b_n}, \tag{6}$$

где

$$K_n = \frac{-c_{n-1}}{a_{n-1}K_{n-1} + b_{n-1}}, \quad L_n = \frac{-a_{n-1}L_{n-1} + f_{n-1}}{a_{n-1}K_{n-1} + b_{n-1}}.$$

Итак, сначала вычисляем коэффициенты L_j, K_j по формуле (4), а затем, начиная с (6), где число y_n знаем, в обратном порядке вычисляем y_j, $j = n-1, n-2, \ldots, 1$.

Параллельный алгоритм (два потока)

Рассмотрим теперь алгоритм решения системы линейных алгебраических уравнений с реализацией выполнения программы на двух потоках. Для наглядности опять возьмем систему из шести уравнений

$$a_k y_{k-1} + b_k y_k + c_k y_{k+1} = f_k, \quad k = 1, 2, \ldots, 6,$$

$$y_0 = 0, \quad y_7 = 0.$$

Выпишем подробно формулы по шагам на каждом потоке.

1 шаг. Из уравнения $b_1 y_1 + c_1 y_2 = f_1$ получаем

$$y_1 = K_2 y_2 + L_2, \quad K_2 = -\frac{c_1}{b_1}, \quad L_2 = \frac{f_1}{b_1}.$$

На первом потоке вычисляем K_2, L_2.

Из последнего уравнения $a_6 y_5 + c_6 y_6 = f_6$ находим

$$y_6 = \widetilde{K}_5 y_5 + \widetilde{L}_5, \quad \widetilde{K}_5 = -\frac{a_6}{b_6}, \quad \widetilde{L}_5 = -\frac{f_6}{b_6}.$$

На втором потоке вычисляем $\widetilde{K}_5, \widetilde{L}_5$.

2 шаг. Преобразуем второе уравнение

$$y_2 = K_3 y_3 + L_3,$$

где

$$K_3 = \frac{-c_2}{a_2 K_2 + b_2}, \quad L_3 = \frac{-a_2 L_2 + f_2}{a_2 K_2 + b_2}.$$

На первом потоке вычисляем K_3, L_3.

На втором потоке вычисляем $\widetilde{K}_4, \widetilde{L}_4$ в соответствии с преобразованиями

$$y_5 = \widetilde{K}_4 y_4 + \widetilde{L}_4, \quad \widetilde{K}_4 = \frac{-a_5}{c_5 \widetilde{K}_5 + b_5}, \quad \widetilde{L}_4 = \frac{f_5 - c_5 \widetilde{L}_5}{c_5 \widetilde{K}_5 + b_5}.$$

3 шаг. Преобразуем третье уравнение

$$y_3 = K_4 y_4 + L_4,$$

где

$$K_4 = \frac{-c_3}{a_3K_3 + b_3}, \quad L_4 = \frac{-a_3L_3 + f_3}{a_3K_3 + b_3}.$$

На первом потоке вычисляем K_4, L_4.

Подготавливаем данные для второго потока.

$$y_4 = \widetilde{K}_3 y_3 + \widetilde{L}_3, \quad \widetilde{K}_3 = \frac{-a_4}{c_4\widetilde{K}_4 + b_4}, \quad \widetilde{L}_3 = \frac{f_4 - c_4\widetilde{L}_4}{c_4\widetilde{K}_4 + b_4}.$$

На втором потоке вычисляем $\widetilde{K}_3, \widetilde{L}_3$

4 шаг. Решаем систему уравнений

$$y_3 = K_4 y_4 + L_4,$$

$$y_4 = \widetilde{K}_3 y_3 + \widetilde{L}_3,$$

получаем

$$y_3 = \frac{L_4 + K_4\widetilde{L}_3}{1 - K_4\widetilde{K}_3}$$

З а д а ч и

1. Составить программу последовательной и параллельной (на два процессора) реализации метода прогонки. При распараллеливании в OPEN MP использовать прагму sections.

```
#pragma omp parallel sections
    { #pragma omp section
      {
         X_calculation();
      }
     #pragma omp section
      {
          Y_calculation();
      }
    }
}
```

Конструкция распределения заданий *sections* предоставляет возможность выполнять каждому потоку свой структурированный блок. Следует отметить, что заранее не известно, какая секция первой выполнит предназначенную ей работу. Запустите на выполнение следующую программу.

```
#include "stdafx.h"
#include <stdio.h>
#include <omp.h>

int _tmain(int argc, _TCHAR* argv[])
{
int n;
  #pragma omp parallel private(n)
     {
     n = omp_get_thread_num();
     #pragma omp sections
     {
     #pragma omp section
        {
         printf("first thread %d\n", n);
        }
     #pragma omp section
        {
         printf("second  thread,  %d\n", n);
        }
     #pragma omp section
         {
          printf(" third thread %d\n", n);
         }
       }
     printf(" end parallel region %d\n", n);
    }
}
```

2. Применить разработанную программу для решения системы уравнений при построении кубического B-сплайна.

Рассмотрим решение систем линейных уравнений методом простых итераций и методом Зейделя.

4.2. Метод простых итераций

Приводим систему уравнений $AX = F$ к виду $X = BX + C$ так, чтобы выполнялось условия теоремы о сходимости метода. Выбираем произвольный начальный вектор $X^{(0)}$. Вычисляем $(n+1)$-е приближение к вектору X по формуле

$$X^{(n+1)} = BX^{(n)} + C.$$

Теорема. Для сходимости метода последовательных приближений необходимо и достаточно, чтобы модуль наибольшего собственного значения матрицы B был меньше единицы, т.е. $|\lambda^B_{\max}| < 1$.

4.2.1. Вычисление наибольшего собственного числа матрицы

Предположим, что наибольшее собственное число $|\lambda_{max}| = |\lambda_1|$ квадратной матрицы B, размера $n \times n$ вещественное и такое, что $|\lambda_{max}| > |\lambda_2| \geq |\lambda_3| \geq \ldots$.

Возьмем произвольный вектор $Y^0 = (y_1, \ldots, y_n)$. Последовательно найдем $Y^1 = BY^0$, $Y^2 = BY^1$, ..., $Y^{k+1} = BY^k$. Известно, что $p_i = Y_i^{k+1}/Y_i^k \to \lambda_{max}$ при $k \to \infty$. Если в мантиссе чисел p_i, $i = 1, \ldots, n$ совпадают s цифр после десятичной точки, то можно считать, что нашли значение для максимального собственного числа с точностью до s-го знака. Для избежания роста компонент вектора Y^k рекомендуется периодически нормировать этот вектор (например, делить на первую или наибольшую по модулю компоненту этого вектора, или приводить вектор к единичной длине).

З а д а ч и

1. Найти наибольшее собственное число матрицы B, $n = 10$,

1000, 1000000, 1000000000:

а) $B = \begin{pmatrix} 0 & 1/4 & 1/8 & 1/16 & \dots & 1/2^{n+1} \\ 1/4 & 0 & 1/4 & 1/8 & \dots & 1/2^{n} \\ 1/8 & 1/4 & 0 & 1/4 & \dots & 1/2^{n-1} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1/2^{n+1} & 1/2^{n} & 1/2^{n-1} & 1/2^{n-2} & \dots & 0 \end{pmatrix}$,

б) $B = \begin{pmatrix} 0 & 1/3 & 1/4 & 1/5 & \dots & 1/(n+1) \\ 1/3 & 0 & 1/5 & 1/6 & \dots & 1/(n+2) \\ 1/4 & 1/5 & 0 & 1/7 & \dots & 1/(n+3) \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1/(n+1) & 1/(n) & 1/(n-1) & 1/(n-2) & \dots & 0 \end{pmatrix}$,

в) $B = \begin{pmatrix} 1 & 2 & 3 & 4 & \dots & n \\ 1 & 1 & 2 & 3 & \dots & n-1 \\ 1 & b & 1 & 2 & \dots & n-2 \\ 1 & b & b & 1 & \dots & n-3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & b & b & b & \dots & 1 \end{pmatrix}$, где $b = (1 - 1./n)$,

Если $|\lambda_{max}| < 1$, решить систему уравнений $X = BX + F$, $F = (1, \dots, 1)$ методом простой итерации.

2. Распараллелить вычисления на два и четыре процессора, а) распараллеливая цикл (при этом матрицу $B = (a)_{ij}^{n}$ можно разбить на две и более прямоугольных матриц например так: $i = 1, \dots, n/2$, $i = n/2 + 1, \dots, n/,$), б) используя параллельные секции (в этом случае разбить матрицу B на две или четыре прямоугольных матрицы). Рассмотреть два случая задания матрицы: массивом B, а также, если это возможно, используя формулы для вычисления элементов массива непосредственно при вычислениях (не выделяя память для хранения массива). Вычислить ускорение и эффективность в каждом случае.

3. При вычислении наибольшего собственного значения матрицы можно использовать перемножение матриц (точнее возведение матрицы в степень $Y^{k+1} = B^{k+1}Y^0$, при этом можно использовать схему сдваивания).

В настоящее время используются несколько схем перемножения матриц.

Рассмотрим две, наиболее употребительные, схемы перемножения квадратных матриц (см.[9]).

Пусть A, B, C — квадратные матрицы $n \times n$, $C = AB$. Компоненты матрицы C обычно рассчитываются по следующей формуле

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}, \quad i, j = 1 \ldots, n.$$

Стандартный порядок следования циклов

```
do i=1,n
  do j=1,n
    do k=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
     end do
    end do
   end do
```

соответствует этой формуле.

На Фортране элементы матрицы располагаются в памяти последовательно по столбцам

$$a_{11}, a_{21}, \ldots, a_{n1}, a_{12}, a_{22}, \ldots, a_{n2}, a_{13}, \ldots, a_{nn}$$

При работе на Фортране более эффективен (по быстродействию) следующий порядок следования циклов

```
do j=1,n
  do k=1,n
    do i=1,n
      c(i,j)=c(i,j)+a(i,k)*b(k,j)
     end do
    end do
   end do
```

Цикл kij предпочтительнее при строковом способе хранения двумерных массивов в оперативной памяти т.е. при программировании на Си.

Другим подходом повышения быстродействия умножения матриц является использование блочного представления матриц

$$C = \begin{pmatrix} C_{11} & \dots & C_{1N} \\ \dots & \dots & \dots \\ C_{N1} & \dots & C_{NN} \end{pmatrix} = AB =$$

$$= \begin{pmatrix} A_{11} & \dots & A_{1N} \\ \dots & \dots & \dots \\ A_{N1} & \dots & A_{NN} \end{pmatrix} \begin{pmatrix} B_{11} & \dots & B_{1N} \\ \dots & \dots & \dots \\ B_{N1} & \dots & B_{NN} \end{pmatrix},$$

$$C_{ij} = \sum_{k=1}^{N} A_{ik} B_{kj},$$

здесь A_{ik}, B_{kj}, C_{ij} – матрицы размера $(n/N) \times (n/N)$, N — количество блоков по строкам или столбцам.

Представим фрагмент программы, реализующей блочное умножение матриц. Пусть $m = n/N$

```
do j=1,N
 do k=1,N
   do i=1,N
     do i1=1,m
       do j1=1,m
         do k1=1,m
 C[i1,j1,i,j]=C[i1,j1,i,j]+A[i1,k1,i,k]B[k1,j1,k,j]
          end do
         end do
        end do
       end do
      end do
     end do
```

4.3. Метод Зейделя

Представим матрицу B в виде $B = M + N$, где

$$M = \begin{pmatrix} 0 & 0 & \dots & 0 & 0 \\ b_{21} & 0 & \dots & 0 & 0 \\ b_{n1} & b_{n2} & \dots & b_{n-1n} & 0 \end{pmatrix},$$

$$N = \begin{pmatrix} b_{11} & b_{12} & \dots & b_{1n-1} & b_{1n} \\ 0 & b_{22} & \dots & b_{2n-1} & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & b_{nn} \end{pmatrix}.$$

Метод Зейделя можно представить в виде

$$X^{(n+1)} = MX^{(n+1)} + NX^{(n)} + C.$$

Отсюда

$$X^{(n+1)} = (I - M)^{(-1)}NX^{(n)} + (I - M)^{(-1)}C.$$

Обозначим $S = (I-M)^{(-1)}N$. Отсюда для сходимости метода необходимо и достаточно, чтобы $|\lambda^S_{\max}| < 1$. λ можно найти из уравнения

$$|(I - M)^{(-1)}N - \lambda I| = 0,$$

которое преобразуем к виду

$$|N - \lambda(I - M)| = 0.$$

Итак, находим λ из уравнения

$$\begin{vmatrix} b_{11} - \lambda & b_{12} & \dots & b_{1n-1} & b_{1n} \\ \lambda b_{21} & b_{22} - \lambda & \dots & b_{2n-1} & b_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ \lambda b_{n1} & \lambda b_{n2} & \dots & \lambda b_{nn-1} & b_{nn} - \lambda \end{vmatrix} = 0$$

и используем теорему о сходимости метода последовательных приближений.

З а д а ч и

1. Применяя MAPLE исследовать на возможность применения метода простой итерации и метода Зейделя для следующих систем уравнений $X = BX + C$, где

$$B = \begin{pmatrix} p & q \\ -q & p \end{pmatrix},$$

при следующих значениях параметров

1) $p = 0,5$, $q = -1$, $C = (1,5,\ -0,5)^T$;
2) $p = -0,9$, $q = -0,3$, $C = (2,2,\ 1,6)^T$;
3) $p = 0,7$, $q = 1,3$, $C = (-1,\ 1,6)^T$;
4) $p = -0,5$, $q = -0,6$, $C = (2,1,\ 1,1)^T$;
5) $p = 0.9$, $q = 0.9$, $C = (-0.8,1)^T$;
6) $p = 0.8$, $q = 1$, $C = (-0.8,1.2)^T$.

4.4. Решение систем линейных уравнений методом Гаусса

Пусть дана система уравнений $AX = F$:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= f_1, \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= f_2, \\ \ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n &= f_n. \end{aligned} \tag{1}$$

1) Прямой ход метода Гаусса

1-й шаг. Допустим, что $a_{11} \neq 0$. Разделим коэффициенты и правую часть первого уравнения на коэффициент a_{11}, который будем называть ведущим. Получаем уравнение

$$x_1 + \overline{a}_{12}^{(0)}x_2 + \ldots + \overline{a}_{1n}^{(0)}x_n = \overline{f}_1^{(0)}.$$

Вычтем из всех уравнений, начиная со второго, это уравнение, умноженное на a_{21} , a_{31} , ..., a_{n1} соответственно. Получаем новые уравнения:

$$a_{22}^{(1)}x_2 + \ldots + a_{2n}^{(1)}x_n = f_2^{(1)},$$

$$\dots\dots\dots\dots\dots \tag{2}$$
$$a_{n2}^{(1)}x_2 + \dots + a_{nn}^{(1)}x_n = f_n^{(1)}.$$

2-й шаг. Разделим коэффициенты первого из преобразованных уравнений на a_{22}, ведущий элемент второго шага, который будем считать отличным от нуля. Получаем уравнение

$$x_2 + \overline{a}_{23}^{(1)}x_2 + \dots + \overline{a}_{2n}^{(1)}x_n = \overline{f}_2^{(1)}.$$

Исключаем x_2 из всех уравнений системы (2), начиная со второго, и продолжим процесс по этой же схеме.

2) Обратный ход. Объединив все первые уравнения каждого шага, получаем систему

$$\begin{gathered} x_1 + \overline{a}_{12}^{(0)}x_2 + \dots + \overline{a}_{1n}^{(0)}x_n = \overline{f}_1^{(0)}, \\ x_2 + \overline{a}_{23}^{(1)}x_3 + \dots + \overline{a}_{2n}^{(1)}x_n = \overline{f}_2^{(1)}, \\ \dots\dots\dots \\ x_n = \overline{f}_n^{(n-1)}. \end{gathered} \tag{3}$$

Из системы (3) последовательно находим x_{n-1}, …, x_1.

Компактная схема метода Гаусса (см. [11]). Обозначим $a_{ij}^{(j-1)}$ через c_{ij}, $i \geq j$. Тогда

$$c_{ij} = a_{ij}^{(j-1)} = a_{ij} - \sum_{l=1}^{j-1} c_{il}b_{lj}, \quad i \geq j,$$

$$b_{ij} = a_{ij}^{(j-1)}/a_{ii}^{(j-1)} = \frac{a_{ij} - \sum_{l=1}^{i-1} c_{il}b_{lj}}{c_{ii}}, \quad i < j.$$

Таким образом, вычисляем элементы b_{ij} прямого хода метода Гаусса,

$$\begin{gathered} c_{11}^1 \; b_{12}^1 \; b_{13}^1 \; b_{14}^1 \\ c_{21}^1 \; c_{22}^2 \; b_{23}^2 \; b_{24}^2 \\ c_{31}^1 \; c_{32}^2 \; c_{33}^3 \; b_{34}^3 \end{gathered}$$

$$c_{41}^1\ c_{42}^2\ c_{43}^3 \dots$$

Элементы с верхним индексом i вычисляются на i-ом шаге.

Пример (см. [10]). Будем решать следующую систему, выполняя арифметические операции с числами типа *Real*:

$$10^{-14}x_1 + 1,00x_2 = 1,00$$

$$1,00x_1 + 1,00x_2 = 2,00.$$

а) Результат применения метода исключения Гаусса без перестановок:

$$x_2 = 1.0, x_1 = 0.0!!!$$

б) Результат применения метода Гаусса с перестановками

$$x_2 = 1,0, \quad x_1 = 1,0.$$

Этот простой пример показывает, что в качестве ведущего элемента следует выбирать максимальный по абсолютному значению коэффициент матрицы уравнений на каждом шаге (стратегия полного упорядочивания) или максимальный по абсолютной величине коэффициент в соответствующем столбце (стратегия частичного упорядочивания Уилкинсона).

в) Результат применения метода Гаусса без перестановок при работе с числами типа *Double*

$$x_2 = 1,0, \quad x_1 = 1,0.$$

Схему единственного деления можно использовать для нахождения обратной матрицы. Пусть

$$A = \{a_{ij}\}_1^n,\ A^{-1} = \{x_{ij}\}_1^n,\ AA^{-1} = E,$$

$$\sum_{k=1}^{n} a_{ik}x_{kj} = \delta_{ij},\ i,j = 1,\dots,n,\ \delta_{ij} = \{\,1\,, i = j, 0, i \neq j.$$

Полученные n систем линейных алгебраических уравнений для $j = 1, \ldots, n$ имеющие одну и ту же матрицу A и различные свободные члены можно одновременно решать методом Гаусса.

Определение. Число обусловленности матрицы A определяется соотношением

$$\mathrm{cond}(A) = \| A\| \ \| A^{-1}\|,$$

таким образом, число обусловленности зависит от используемой нормы.

З а д а ч и

1. а) Вычислить норму матрицы B (из раздела по решению системы линейных уравнений методом простой итерации) по формулам

$$\|B\| = \max_i \sum_j |b_{ij}|, \ \|B\| = \max_j \sum_i |b_{ij}|.$$

Предложить схемы распараллеливания вычислений при размере матрицы $n > 10000000$. Предложить параллельный вариант вычислений при одновременном использовании обеих формул.

Вначале возьмите систему уравнений при небольшом n, например, $n = 4$, убедитесь, что параллельный и последовательный алгоритмы дают одинаковый (правильный) результат. Затем следует вычислить время выполнения и ускорение при распараллеливании вычислений для систем уравнений при произвольном n, например $n = 10000000$. Не забудьте, что для организации параллельных потоков потребуется дополнительное время, равное примерно 2000 длинных операций (умножений и делений).

```
  #include "stdafx.h"
  #include <stdio.h>
  #include <omp.h>

int _tmain(int argc, _TCHAR* argv[])
{

  #pragma omp parallel sections
```

```
{
   #pragma omp section
   {
       ......................
   }
#pragma omp section
   {
    ..................
     }
  }
}
```

б) Составить программу для вычисления B^{-1}. Предложить параллельный вариант программы для вычисления B^{-1}.

2. Решить систему уравнений при $n = 10, 20, 40, 80, 200$.

$$1 + x_1 + x_2 + x_3 + \ldots + x_n = 0,$$

$$1 + 2x_1 + 2^2x_2 + 2^3x_3 + \ldots + 2^nx_n = 0,$$

$$\ldots \quad \ldots \quad \ldots \quad \ldots \quad \ldots$$

$$1 + nx_1 + n^2x_2 + n^3x_3 + \ldots + n^nx_n = 0.$$

4.5. О влиянии числа обусловленности

Рассмотрим пример (см. [11]):

$$A = \begin{pmatrix} 1 & 0.99 \\ 0.99 & 0.98 \end{pmatrix}.$$

Находим собственные числа матрицы A $\lambda_2 = -0.00005$, $\lambda_1 = 1.98005$, так как матрица $A = A^T$, то матрица $AA^T = A^2$ имеет собственные значения $(0.00005)^2$, $(1.98005)^2$, тогда $\mu_2 = 0.00005$, $\mu_1 = 1.98005$, $\mathrm{cond}(A) = 39600$.

Рассмотрим систему

$$x_1 + 0.99x_2 = 1.99,$$

$$0.99x_1 + 0.98x_2 = 1.97.$$

Точное решение этой системы $x_1 = 1$, $x_2 = 1$. Однако $x_1 = 3$, $x_2 = -1.0203$ являются решением системы

$$x_1 + 0.99x_2 = 1.989903,$$

$$0.99x_1 + 0.98x_2 = 1.970106,$$

т.е. изменение

$$\delta b = \begin{pmatrix} -0.000097 \\ 0.000106 \end{pmatrix}$$

дает

$$\delta x = \begin{pmatrix} 2.0 \\ -2.0203 \end{pmatrix}.$$

З а д а ч и

1. Вычислить число обусловленности матрицы Гильберта H_n с элементами $\{h_{ij}\}_{ij=1}^n$, где $h_{ij} = \frac{1}{i+j-1}$ при $n = 10, 20, 40, 80$.

4.6. Решение систем линейных уравнений методом квадратного корня

В случае, если матрица системы симметрична, нахождение решения можно упростить. В случае конечно-разностной аппроксимации обыкновенных дифференциальных уравнений и решения краевых задач вариационными методами часто приходится решать систему уравнений с ленточной симметричной матрицей. Если матрица системы уравнений положительно определенная, то предлагаемый процесс не имеет осложнений (см. [11]). предположим, что элементы матрицы хранятся в памяти последовательно по строкам.

Разложим матрицу A в произведение двух транспонированных друг другу треугольных матриц

$$A = S^T S,$$

где

$$S = \begin{pmatrix} s_{11} & s_{12} & \dots & s_{1n-1} & s_{1n} \\ 0 & s_{22} & \dots & s_{2n-1} & s_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 & s_{nn} \end{pmatrix}.$$

В силу правила умножения матриц получаем элементы s_{ij}:

$$s_{11} = \sqrt{a_{11}}, \quad s_{1j} = \frac{a_{1j}}{s_{11}}, \qquad s_{ii} = \sqrt{a_{ii} - \sum_{l=1}^{i-1} s_{li}^2}, \quad (i > 1),$$

$$s_{ij} = \frac{a_{ij} - \sum_{l=1}^{i-1} s_{li} s_{lj}}{s_{ii}}, \quad (j > i), \quad s_{ij} = 0, \quad (i > j).$$

Далее решаем две рекуррентные системы

$$S'Z = F, \quad SX = Z.$$

Имеем

$$z_1 = \frac{f_1}{s_{11}}, \quad z_i = \frac{f_i - \sum_{l=1}^{i-1} s_{li} z_l}{s_{ii}}, \quad (i > 1).$$

Обратный ход осуществляется по формулам

$$x_n = \frac{z_n}{s_{nn}}, \quad x_i = \frac{z_i - \sum_{l=i+1}^{n} s_{il} x_l}{s_{ii}}, \quad (i = n-1, \dots, 1).$$

Подсчитаем число операций для выполнения разложения. Вычисления по формулам

$$s_{11} = \sqrt{a_{11}}, \qquad s_{ii} = \sqrt{a_{ii} - \sum_{l=1}^{i-1} s_{li}^2}, \quad (i > 1),$$

требуют

$$\sum_{l=2}^{n} 2(l-1) = n(n-1)$$

операций. Вычисления по формулам

$$s_{ij} = \frac{a_{ij} - \sum_{l=1}^{i-1} s_{li} s_{lj}}{s_{ii}}, \quad (j > i), \quad s_{ij} = 0, \quad (j > i).$$

при каждом фиксированном i требуют

$$\sum_{l=2}^{i-1} 2(l-1) = (i-2)(i-1)$$

операций. Всего

$$\sum_{i=2}^{n} (i-2)(i-1) = n(n-1)(n-2)/3.$$

и n операций извлечения квадратного корня.

4.7. Решение систем линейных уравнений методом квадратного корня (второй вариант)

Пусть элементы матрицы хранятся в памяти последовательно по столбцам. Разложим действительную симметричную положительно определенную матрицу A в произведение двух транспонированных друг другу треугольных матриц

$$A = SS^T,$$

где S —нижняя треугольная матрица с положительными элементами на главной диагонали. Элементы матрицы S можно вычислить, начиная с ее левого угла по следующим формулам

$$s_{11} = \sqrt{a_{11}}, \quad s_{i1} = \frac{a_{i1}}{s_{11}}, \qquad s_{ii} = \sqrt{a_{ii} - \sum_{l=1}^{i-1} s_{il}^2}, \quad (i > 1),$$

$$s_{ij} = \frac{a_{ij} - \sum_{l=1}^{j-1} s_{il} s_{jl}}{s_{jj}}, \quad i = 2, \ldots, n, i = 2, \ldots, i-1.$$

Далее решаем две рекуррентные системы

$$SZ = F, \quad S^T X = Z.$$

Имеем

$$z_1 = \frac{f_1}{s_{11}}, \quad z_i = \frac{f_i - \sum_{l=1}^{i-1} s_{il} z_l}{s_{ii}}, \quad (i > 1).$$

Обратный ход осуществляется по формулам

$$x_n = \frac{z_n}{s_{nn}}, \quad x_i = \frac{z_i - \sum_{l=i+1}^{n} s_{li} x_l}{s_{ii}}, \quad (i = n-1, \ldots, 1).$$

Блочный алгоритм

Пусть r — размер блока. Представим исходную матрицу A в виде

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix}$$

Размер A_{11} $r \times r$, A_{21} $(n-r) \times r$, A_{22} $(n-r) \times (n-r)$.

Искомую матрицу S запишем в виде

$$S = \begin{pmatrix} S_{11} & 0 \\ S_{21} & S_{22} \end{pmatrix}$$

Ввиду соотношения

$$A = SS^T$$

имеем

$$A = \begin{pmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} S_{11} & 0 \\ S_{21} & S_{22} \end{pmatrix} \begin{pmatrix} S_{11}^T & S_{21}^T \\ 0 & S_{22}^T \end{pmatrix} =$$

$$\begin{pmatrix} S_{11}S_{11}^T & S_{11}S_{21}^T \\ S_{21}S_{11}^T & S_{21}S_{21}^T + S_{22}S_{22}^T \end{pmatrix}.$$

Отсюда получаем

$$A_{11} = S_{11}S_{11}^T,$$

$$A_{21} = S_{21}S_{11}^T,$$
$$A_{22} = S_{21}S_{21}^T + S_{22}S_{22}^T,$$

Блок S_{11} находим с помощью обычного алгоритма.

Далее используя известный блок A_{21} и найденный блок S_{11} находим блок A_{21} Для этого нужно решить r вспомогательных систем уравнений. Ввиду того, что матрица L_{11} треугольная, то для решения одной системы требуется $O(r^2)$ операций, а всего для нахождения L_{21} требуется $O(r^3)$.

Далее находим

$$B_{22} = A_{22} - S_{21}S_{21}^T = S_{22}S_{22}^T.$$

З а д а ч и

Решить систему уравнений методом квадратного корня

$$\begin{pmatrix} n & 1/n & 1/(2n) & 1/(3n) & \dots & \frac{1}{(n-1)n} \\ 1/n & 2n & 1/n & 1/(2n) & \dots & \frac{1}{(n-2)n} \\ 1/(2n) & 1/n & 3n & 1/n & \dots & \frac{1}{(n-3)n} \\ 1/(3n) & 1/(2n) & 1/n & 4n & \dots & \frac{1}{(n-4)n} \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \frac{1}{(n-1)n} & \frac{1}{(n-2)n} & \frac{1}{(n-3)n} & \frac{1}{(n-4)n} & \dots & nn \end{pmatrix}$$

5. ПОСТРОЕНИЕ КВАДРАТУРНЫХ ФОРМУЛ

5.1. Составная квадратурная формула прямоугольников

Разделим промежуток $[a, b]$ на n частичных промежутков длины h, $h = (b-a)/n$, точками деления $x_k = a + kh$, $k = 0, 1, \ldots, n$. Применяя к каждому частичному промежутку $[x_k, x_{k+1}]$ формулу прямоугольников и суммируя по k от 0 до $n-1$, получаем

$$\int_a^b f(x)dx \approx h \sum_{k=0}^{n-1} f(\alpha + kh),$$

где $\alpha \in [a, a+h]$; при $\alpha = a$, $\alpha = a+h$, $\alpha = a + h/2$ получаем соответственно составную формулу левых, правых и средних прямоугольников.

Замечание. Квадратурная формула

$$\int_0^{2\pi} f(x)dx \approx \frac{2\pi}{n} \sum_{k=1}^{n} f\left(\alpha + (k-1)\frac{2\pi}{n}\right)$$

точна для всех тригонометрических полиномов порядка $n-1$. Здесь $\alpha \in [0, 2\pi/n]$.

З а д а ч а

1а) В среде Maple подключить библиотеку командой *with* (*student*). Рассмотреть иллюстрацию построения квадратурных формул левых, правых, средних прямоугольников на примере $f(x) = x^2$, $a = 0$, $b = 1$ командами $leftbox$, $rightbox$, $middlebox$. Например, $leftbox$ $(x * x,\ x = 0..1,\ 7)$. Цифра "7"задает число разбиений, по умолчанию число разбиений 5.

При $c = 0.9$, $c = 0.1$ вычислить

$$\int_0^{\pi} \ln(1 + 2c\cos(x) + c^2)dx$$

с помощью команд $rightsum, middlesum, leftsum$.

1б) Составить программу вычисления интеграла по составной формуле средних прямоугольников.

Возможное решение:

$f := unapply(x^3, x);\ a := 0;\ n := 100; s := evalf(h * sum(f(a + h/2 + k * h),\ k = 0..n - 1),\ 50);$

число 50 означает, что на печать выводится 50 цифр.

2. Показать, что для коэффициентов интерполяционной квадратурной формулы имеет место равенство

$$\sum_{k=1}^{n} A_k = \int_a^b p(x)dx.$$

3. Привести примеры интерполяционных квадратурных формул с положительными узлами.

4. Показать, что интерполяционная квадратурная формула с n узлами точна для всех полиномов степени не выше $n - 1$.

5.2. Квадратурная формула Ньютона—Котеса. Составные квадратурные формулы

5.2.1. Квадратурная формула Ньютона—Котеса

Коэффициенты квадратурной формулы Ньютона—Котеса

$$\int_a^b f(x)dx \approx (b - a) \sum_{k=0}^{n} B_k f(x_k)$$

задаются формулами

$$B_k = \frac{(-1)^{n-k}}{k!(n-k)!n} \int_0^n \frac{t(t-1)\dots(t-n)}{t-k} dt,$$

а $x_k = a + kh$, $k = 0, 1, \dots, n$, $h = (b - a)/n$.

З а д а ч и

а) Вычислить коэффициенты квадратурной формулы при $n = 3, 4, 5$. Задать подынтегральную функцию процедурой-функцией с помощью команды *unapply*.

б) Вычислить

$$\int_0^{\pi} \sin(3x)dx$$

по составной квадратурной формуле прямоугольников и составной формуле трапеций при $n = 5$. Объяснить результаты.

в) Вычислить интеграл по формуле Ньютона—Котеса при $n > 10$. Исследовать погрешность, связанную с округлением. Улучшить результаты вычислений, увеличив количество знаков в мантиссе (команды $Digits := 20;\ Digits := 30;$ и т.д.).

Указание. Ввиду того, что нумерация элементов массива начинается с 1, преобразуем приведенные выше формулы

$$\int_a^b f(x)dx \approx (b-a)\sum_{k=1}^{n+1} B_k f(x_k),$$

где

$$B_k = \frac{(-1)^{n+1-k}}{(k-1)!(n+1-k)!\ n}\int_0^n \frac{t(t-1)\dots(t-n)}{t-k+1}dt.$$

Для вычисления произведения используем команду *product*.

5.2.2. Составные квадратурные формулы трапеций и Симпсона

Составная формула трапеций имеет вид

$$\int_a^b f(x)dx \approx \frac{b-a}{2n}\Big(f_0 + 2(f_1 + \ldots + f_{n-1}) + f_n\Big).$$

Составная формула Симпсона при четном n:

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}\Big(f_0 + 4(f_1 + f_3 + \ldots + f_{n-1}) + \\ +2(f_2 + f_4 + \ldots + f_{n-2}) + f_n\Big).$$

З а д а ч а

1. Разработать параллельный вариант составной формулы средних прямоугольников. Вычислить следующие интегралы

$$\int_0^{\pi/2} \frac{\sin^2 x dx}{(a^2 \sin^2 x + b^2 \cos^2 x)^2},\ \int_0^{\pi/2} \frac{\cos^2 x dx}{(a^2 \sin^2 x + b^2 \cos^2 x)^2},\ a = \frac{1}{4}, b = \frac{3}{2}.$$

2. Указать случаи, когда квадратурная формула трапеций дает значение интеграла с недостатком, а когда с избытком.

3. Оценить погрешность при вычислении следующих интегралов по формуле средних прямоугольников и трапеций $\int_1^2 \frac{dx}{x},\ n = 5,$ $\int_0^2 \frac{dx}{x+1},\ n = 5, \int_0^1 \frac{dx}{x^2+1},\ n = 5$. Составить программу, которая вычисляет одновременно интегралы по формуле средних прямоугольников и трапеций. Вычислить интегралы

$$\int_0^{\pi} \frac{x \sin mx dx}{(1 + \cos^2 mx)^2},\ m = \frac{1}{4},\ \frac{3}{2}.$$

4. Оценить погрешность вычисления интегралов по составной формуле Симпсона $\int_1^2 \frac{dx}{x+1},\ n = 5, \int_0^2 \frac{dx}{x^3+1},\ n = 5, \int_0^1 \frac{xdx}{x+1},\ n = 5.$ Предложить параллельный вариант вычисления интегралов по составной формуле Симпсона и вычислить интегралы

$$\int_0^{\pi} \frac{\sin x dx}{\sqrt{a^2 - 2ab\cos(x) + b^2}},\ a = \frac{1}{4}, b = \frac{3}{2}, \int_0^{\pi} \frac{\sin x \sin 5x dx}{1 - 2a\cos(x) + a^2},\ a = \frac{1}{4}.$$

5. Вычислить интеграл по формуле Эйлера-Маклорена

$$\int_a^b f(x)dx = T_n(f) - \sum_{m=2}^{\nu-1} \frac{h^m B_m}{m!} \left(f^{(m-1)}(b) - f^{(m-1)}(a) \right) +$$

$$\frac{h^{\nu+1}}{\nu!}\int_0^1 (B_\nu(1-\tau)-B_\nu)\sum_{j=0}^{n-1} f^{(\nu)}(a+jh+h\tau)d\tau,$$

где

$$T_n(f)=h\biggl(\frac{f(a)}{2}+\sum_{r=1}^{n-1} f(a+rh)+\frac{f(b)}{2}\biggr),$$

числа Бернулли B_k определяем из рекуррентных соотношений

$$B_0=1,\ \frac{B_0}{k!}+\frac{B_1}{(k-1)!1!}+\frac{B_2}{(k-2)!2!}+\ldots+\frac{B_{k-1}}{1!(k-1)!}=0,$$

$k=2,3,4,\ldots.$ $B_0=1$, $B_1=-1/2$, $B_2=1/6$, $B_3=0$, $B_4=-1/30$,

Пусть $a=x_0,x_1,\ldots<x_n$. Представим интеграл

$$\int_a^b f(x)dx=\sum_{i=0}^{n-1}\int_{x_k}^{x_{k+1}} f(x)dx.$$

Приведем линейным преобразованием интеграл $\int_{x_k}^{x_{k+1}} f(x)dx$ к виду $\int_0^1 f(t)dt$. Пусть $t_i=a+ih$, $h=(b-a)/m$, $f_i=f(i/m)$, $f'_i=f'(i/m)$. Будем применять формулы вида

$$\int_0^1 p(t)f(t)dt\approx\sum_{i=0}^{m}\biggl(A_{i0}f_i+A_{i1}f'_i\biggr).$$

В случае постоянной весовой функции

$$m=1,\ \int_0^1 f(t)dt=\frac{1}{2}\biggl(f_0+f_1-\frac{1}{6}(f'_1-f'_0)\biggr)+\frac{1}{720}f^{(4)}(\xi),$$

$$m=2,\ \int_0^1 f(t)dt=\frac{1}{30}\biggl(7(f_0+f_1)+16f_1-\frac{1}{2}(f'_2-f'_0)\biggr)+\frac{1}{2^7 4725}f^{(6)}(\xi),$$

$$m=3,\ \int_0^1 f(t)dt=\frac{1}{1120}\biggl(155(f_0+f_3)+405(f_1+f_2)+$$

$$+\frac{1}{3}(27(f_2'-f_1')-19((f_3'-f_0'))\Big)+\frac{1}{3^7 313600}f^{(8)}(\xi),$$

З а д а ч и

1. Вычислить

$$J_1=\int_0^{\pi}\frac{1}{(1+0.9\cos(x))^2}dx.$$

Проверить, что $J_1=\pi/(1-0.9^2)^{3/2}$.

2. Вычислить при $a=0.1$, $b=0.9$

$$J_2=\int_0^{\pi/2}\frac{1}{(a\sin(x)+b\cos(x))^2}dx.$$

Проверить, что $J_2=1/(ab)$.

5.3. Вычисление повторного интеграла

Предположим, что нужно вычислить повторный интеграл

$$J=\iint_{\Omega}f(x,y)dxdy,$$

где область Ω представляет собой прямоугольник $a\le x\le b, c\le y\le d$. В этом случае $J=\int_a^b dx\int_c^d f(x,y)dy$. Обозначим $F(x)=\int_c^d f(x,y)dy$, тогда $J=\int_a^b F(x)dx$. Для вычисления последнего интеграла применим любую составную квадратурную формулу, например, формулу средних прямоугольников или формулу трапеций. $h=(b-a)/n$, $x_k=a+kh$, $k=0,1,\ldots,n$,

$$J=\int_a^b F(x)dx\approx\frac{h}{2}\Big(F_0+2\sum_{i=1}^{n-1}F_i+F_n\Big),$$

$$F_i=F(x_i)=\int_c^d f(x_i,y)dy,\ i=0,1,\ldots,n.$$

З а д а ч и

Вычислить повторный интеграл при $a = 0, b = 1, c = 0, d = 1$, $f(x, y) = 5y^3 \sin(5x)$.

5.4. Квадратурные формулы гауссова типа

5.4.1. Квадратурная формула Гаусса

Квадратурная формула Гаусса

$$\int_{-1}^{1} f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

имеет своими узлами корни многочлена Лежандра

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n (x^2 - 1)^n}{dx^n}.$$

Коэффициенты A_k вычисляются по формуле

$$A_k = \frac{2}{(1 - x_k^2)[P_n'(x_k)]^2}, \quad k = 1, \ldots, n.$$

Приведем узлы и коэффициенты формулы Гаусса для некоторых n:

$$n = 1,$$

$x_1 = 0,\ A_1 = 2;$

$$n = 2,$$

$x_2 = -x_1 = 1/\sqrt{3} \approx 0.5773502692,\ A_1 = A_2 = 1;$

$$n = 3,$$

$x_2 = 0,\ A_2 = 8/9,$
$x_3 = -x_1 = \sqrt{3/5} \approx 0.7745966692,\ A_1 = A_3 = 5/9;$

$$n = 4,$$

$x_3 = -x_2 \approx 0.3399810436,\ A_2 = A_3 \approx 0.6521451549,$
$x_4 = -x_1 \approx 0.8611363116,\ A_4 = A_1 \approx 0.3478548451;$

$n = 5,$

$x_3 = 0,\ A_3 \approx 0.5688888899,$
$x_4 = -x_2 \approx 0.5384693101,\ A_2 = A_4 \approx 0.4786286705,$
$x_5 = -x_1 \approx 0.9061798459,\ A_1 = A_5 \approx 0.2369268851.$

Замечание.

$$\int_a^b f(t)dt = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{b+a}{2}\right)dx,$$

поэтому

$$\int_a^b f(t)dt \approx \frac{b-a}{2}\sum_{k=1}^{n} A_k f(t_k),$$

где

$$t_k = \frac{b-a}{2}x_k + \frac{b+a}{2},$$

x_k — узлы квадратурной формулы Гаусса, $x_k \in [-1, 1]$, A_k — соответствующие им коэффициенты.

З а д а ч а

Вычислить интеграл с помощью квадратурной формулы Гаусса

$$\int_0^1 \frac{dt}{1+t^2} = \pi/4$$

с погрешностью 10^{-5}.

5.4.2. Квадратурная формула Мелера

Квадратурная формула Мелера имеет вид

$$\int_{-1}^{1} \frac{1}{\sqrt{1-x^2}} f(x)dx \approx \pi/n \sum_{k=1}^{n} f(x_k),$$

где

$$x_k = \cos\left(\frac{2k-1}{2n}\pi\right), \quad k = 1, \ldots, n.$$

З а д а ч а

Вычислить интегралы

$$\int_{-1}^{1} \frac{e^{2t}dt}{\sqrt{1-t^2}}, \quad \int_{-1}^{1} \frac{dt}{(1+t^2)\sqrt{1-t^2}}$$

с погрешностью 10^{-5}.

5.4.3. Квадратурная формула Чебышева—Эрмита

Квадратурная формула

$$\int_{-\infty}^{+\infty} e^{-x^2} f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

имеет своими узлами корни многочлена Чебышева—Эрмита

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} e^{-x^2}.$$

Коэффициенты вычисляются по формуле

$$A_k = \frac{2^{n+1} n! \sqrt{\pi}}{\left(H_n'(x_k)\right)^2}.$$

Приведем узлы и коэффициенты формулы для некоторых n:

$$n = 4,$$

$-x_1 = x_4 \approx 1.6506801239$, $A_1 = A_4 \approx 0.08131283545$,
$-x_2 = x_3 \approx 0.5246476233$, $A_2 = A_3 \approx 0.8049140900$;

$$n = 5,$$

$x_3 = 0,\ A_3 \approx 0.9453087205,$
$x_4 = -x_2 \approx 0.9585724646,\ A_2 = A_4 \approx 0.3936193232,$
$x_5 = -x_1 \approx 2.0201828705,\ A_1 = A_5 \approx 0.01995324206.$

З а д а ч а

Вычислить интегралы

$$\int_{-\infty}^{+\infty} \frac{e^{-t^2}dt}{1+t^2},\ \int_{-\infty}^{+\infty} \frac{e^{-t^2}dt}{5+t}$$

при $n = 4, 5$.

5.4.4. Квадратурная формула Чебышева—Лагерра

Квадратурная формула

$$\int_0^{+\infty} x^\alpha e^{-x} f(x)dx \approx \sum_{k=1}^{n} A_k f(x_k)$$

имеет своими узлами корни многочлена Чебышева—Лагерра

$$L_n^{(\alpha)}(x) = (-1)^n x^{-\alpha} e^x \frac{d^n}{dx^n}(x^{\alpha+n}e^{-x}).$$

Коэффициенты вычисляются по формулам

$$A_k = \frac{n!\Gamma(\alpha+n+1)}{x_k((L_n^{(\alpha)}(x_k))')^2}.$$

Приведем узлы и коэффициенты формулы для некоторых n при $\alpha = 0$:

$$n = 4,$$

$x_1 \approx 0.3225476896,\ A_1 \approx 0.6031541043,$
$x_2 \approx 1.7457611012,\ A_2 \approx 0.3574186924,$
$x_3 \approx 4.5366202969,\ A_3 \approx 0.0388879085,$
$x_4 \approx 9.3950709123,\ A_4 \approx 0.0005392947;$

$$n = 5,$$

$x_1 \approx 0.2635603197$, $A_1 \approx 0.5217556106$,
$x_2 \approx 1.4134030591$, $A_2 \approx 0.3986668111$,
$x_3 \approx 3.5964257710$, $A_3 \approx 0.0759424497$,
$x_4 \approx 7.0858100059$, $A_4 \approx 0.0036117558$,
$x_5 \approx 12.6408008443$, $A_5 \approx 0.0000233700$.

З а д а ч а

Вычислить интегралы

$$\int_0^{+\infty} \frac{e^{-t}dt}{2+t}, \quad \int_0^{+\infty} \frac{e^{-t}t\ dt}{1-e^{-2t}}$$

при $n = 4, 5$.

5.4.5. Построение квадратурных формул гауссова типа

Теорема. Для того чтобы квадратурная формула с n узлами была точна на многочленах степени не выше $2n-1$, необходимо и достаточно, чтобы: 1) она была интерполяционная; 2) ее узлы были корнями многочлена $\omega(x) = (x-x_1)\cdot\ldots\cdot(x-x_n)$, такого что

$$\int_a^b p(x)\omega(x)Q_m(x)\,dx = 0$$

для любого многочлена Q_m степени $m = 0, 1, \ldots, n-1$.

З а д а ч а

Построить квадратурную формулу гауссова типа с двумя узлами

$$\int_a^b p(x)f(x)dx \approx A_1 f(x_1) + A_2 f(x_2).$$

Пояснения к решению. Пусть x_1 и x_2 — корни многочлена второй степени $\omega(x)$, т.е. $\omega(x) = (x-x_1)(x-x_2)$. Запишем $Q(x)$ в виде $Q(x) = x^2 + sx + r$. Условия ортогональности $Q(x)$ к многочленам

нулевой и первой степени на данном промежутке по данному весу запишем в виде

$$\int_a^b p(x)(x^2+sx+r)dx=0, \quad \int_a^b p(x)x(x^2+sx+r)dx=0.$$

Использовав определение моментов $c_i=\int_a^b p(x)x^i\,dx$, запишем систему уравнений в виде

$$\begin{cases} c_2+sc_1+rc_0=0, \\ c_3+sc_2+rc_1=0. \end{cases}$$

Итак: 1) вычисляем моменты c_i, $i=0,1,2,3$; 2) решаем систему уравнений и находим неизвестные s,r; 3) находим x_1, x_2; 4) определяем коэффициенты A_1, A_2, например, из условий точности квадратурной формулы на многочленах степени не выше третьей. Условия точности на многочленах нулевой и первой степени дают систему уравнений

$$\begin{cases} A_1+A_2=c_0, \\ A_1x_1+A_2x_2=c_1. \end{cases}$$

Варианты заданий:
а) $p(x)=x^\alpha$, $\alpha=1/2,2,3$, $a=0$, $b=1$;
б) $p(x)=e^x$, $a=0$, $b=0{,}5$;
в) $p(x)=\ln(x)$, $a=0$, $b=1/2$.

Литература

1. *А.С. Антонов.* Параллельное программирование с использованием технологии с использованием технологии OpenMP

2. *Бахвалов Н. С.* Численные методы. М., 1975. 632 с.

3. *Березин И. С., Жидков Н. П.* Методы вычислений. Т. 2. М., 1962. 620 с.

4. *Бурова И. Г., Демьянович Ю. К.* Теория минимальных сплайнов. СПб., 2000. 316 с.

5. *Завьялов Ю. С., Квасов Б. И., Мирошниченко В. Л.* Методы сплайн-функций. М., 1980. 352 с.

6. *Крылов В. И.* Приближенное вычисление интегралов. М., 1967. 500 с.

7. *Матросов А. В.* Maple 6. Решение задач высшей математики и механики. СПб., 2001. 528 с.

8. *Мысовских И. П.* Лекции по методам вычислений. СПб., 1998. 472 с.

9. *Старченко А.В., Данилкин Е.А., Лаева В.И., Проханов С.А.* практикум по методам параллельных вычислений. Изд-во МГУ. 2010. 200 с.

10. *Фаддеев Д. К., Фаддева В. Н.* Вычислительные методы линейной алгебры. М.; Л., 1963. 734 с.

11. *Форсайт Дж., Молер К.* Численное решение систем линейных алгебраических уравнений. М., 1969. 166 с.

ПРИЛОЖЕНИЕ
КРАТКОЕ ЗНАКОМСТВО С MAPLE

Список основных команд с комментариями

Для получения информации об использовании команды и параметрах нужно вызвать справку по следующей схеме: ? название команды.

Любая команда должна заканчиваться знаком ":"или ";причем двоеточие служит для подавления вывода на экран.

В ряде случаев необходимо подключение соответствующей библиотеки, что осуществляется командой *with* (название библиотеки) или *readlib* (название библиотеки).

1. Операции с формулами

normal — сокращение дроби;
numer — выделение числителя рациональной дроби;
denom — выделение знаменателя рациональной дроби;
expand — раскрытие скобок;
subs — подстановка;
simplify — упрощение выражения;
collect — приведение подобных членов относительно указанной переменной;
combine — приведение подобных членов относительно функции;
lhs — выделение левой части уравнения;
rhs — выделение правой части уравнения;
op — извлечение подвыражений первого уровня;
coeff — определение коэффициента при степени переменной;
whattype — определение типа переменной;
convert — преобразование типа;
series — разложение в ряд Тейлора;

$eval$ — просмотр содержимого переменной;
$factor$ — разложение полинома на множители.

2. Операции вычисления

$evalf$ — вычисление выражения с плавающей запятой;
$frac$ — вычисление дробной части;
$round$ — округление действительного переменного;
$trunc$ — вычисление целой части выражения;
Re — определение вещественной части выражения;
Im — определение мнимой части выражения;
max — нахождение максимума;
min — нахождение минимума.

3. Решение уравнений и неравенств

$solve$ — решение уравнений;
$fsolve$ — численное решение уравнений;
$RootOf$ — нахождение корня уравнения.

4. Математический анализ

$limit$ ($Limit$) — вычисление предела;
sum (Sum) — операция суммирования;
$product$ ($Product$) — вычисление произведения;
$extrema$ — вычисление экстремумов;
$iscont$ — исследование непрерывности;
$diff$ ($Diff$) — дифференцирование;
int (Int) — интегрирование.

5. Линейная алгебра

$matrix$, $array$ — задание матриц;
$vector$ — задание вектора;
$linsolve$ — решение системы линейных алгебраических уравнений.

6. Решение дифференциальных уравнений

dsolve — решение дифференциальных уравнений;
DEplot — графическое изображение решений дифференциальных уравнений.

7. Построение графиков

plot — построение двумерных графиков;
*plot*3*d* — построение трехмерных изображений.

8. Программирование

Условный оператор

$if\ \ bool\ \ then\ \ expr1\ \ else\ \ expr2\ \ fi;$
$if\ \ bool1\ \ then\ \ expr1\ \ elif\ \ bool2\ \ then\ \ expr2\ \ ...$
$elif\ \ booln\ \ then\ \ exprn\ \ else\ \ expr0\ \ fi;$

Операторы цикла

$for\ \ var\ \ from\ \ v1\ \ by\ \ v2\ \ to\ \ v3\ \ do\ \ expr\ \ od;$
$while\ \ bool\ \ do\ \ expr\ \ od;$
$for\ \ var\ \ from\ \ v1\ \ by\ \ v2\ \ while\ \ bool\ \ do\ \ expr\ \ od;$
$for\ \ var\ \ in\ \ expr\ \ do\ \ expr2\ \ od;$

Замечание. В некоторых версиях для обозначения конца цикла можно применять *end do*.

Процедуры-функции

$name := (var1, var2, ..,) - > expr;$
$name :=< expr|var1, var2, ... >;$
$name := unapply(expr, var1, var2, ...);$

Процедуры

$name := proc(var1, var2, ...)$
$local...\ \ global...\ \ options...\ \ expr1;\ \ expr2; ...$
$end;$

9. Запись и считывание данных

Для записи данных в файл служит оператор $writedata$:

$writedata[APPEND](fileD, data)$;

$writedata[APPEND](fileD, data, format)$;

здесь $fileD$ — имя файла данных, $data$ — список, вектор или матрица данных, $format$ — спецификация формата данных ($integer$, $float$ или $string$).

Необязательный указатель $APPEND$ используется, если данные должны дописываться в уже созданный файл. Считывание из файла $filename$ обеспечивает функция $readdata$:

$readdata(fileD, n)$;

$readdata(fileD, format, n)$;

здесь n — целое положительное число, задающее число считываемых столбцов.

Пример операции.

$> data := array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) :$
$> writedata('D : \backslash mydata.txt', data) :$
$> restart;$
$> data1 := readdata('D : \backslash mydata.txt', 3) :$

Замечание. Команду $restart$ рекомендуется ставить в начале программы, чтобы при следующем ее запуске переменные принимали исходные символьные значения.

10. Кусочно заданные функции

С функциями типа $piecewise$ можно работать как с обычными функциями.

$> restart;$
$> f := x- > piecewise(x^2 > 4,\ x^2,\ -2 * x^2) :$
$> plot(f(x), x = -5..5);$
$> restart;$
$> f := max(x^2 - 2, x - 1);$
$> g := convert(f, piecewise) :$

$> restart; with(plots); with(linalg);$
$> A := hilbert(8); matrixplot(A);$

Приближенные величины и действия над ними

Пусть x — произвольное действительное число. Если оно рациональное, то его десятичное представление содержит либо конечное число цифр, либо бесконечное (периодическая десятичная дробь). Заметим, что и в первом случае количество цифр может быть сколь угодно большим. Если число иррациональное, то его десятичная запись содержит бесконечное количество цифр. Однако на практике все вычисления, в том числе и на ЭВМ, проводятся с конечным числом разрядов (десятичных цифр) и поэтому данное число округляется до соответствующего количества разрядов.

Например, число $\pi = 3.1415926535897\ldots$ с точностью до трех цифр равно 3.14.

Ограниченность числа разрядов при вычислениях приводит к погрешностям вычислений. Следующие примеры в среде MAPLE иллюстрируют сказанное.

Пример 1. Вычислим сумму $a+b$ и отношение $\dfrac{a+b}{b}$ при $a = 1$, $b = 4.44444444 \cdot 10^{-8}$ при значении параметра $Digits = 8$ и $Digits = 9$.

В приведенном ниже фрагменте приведены результаты вычислений, выполненные в диалоговом режиме.

```
> restart;
> Digits:=8;
> a:=1:   b:=4.44444444*10^(-8);
                b := 0.44444444  10^{-7}
> c:=a+b;
                 1.0000000
> c/b;
                 0.22500000  10^{8}
```

Таким образом, получили результат, равный 22500000.

Пересчитаем теперь при другом значении параметра $Digits$.

```
> Digits:=9;
> a:=1: b:=4.44444444*10^(-8);
```

```
                          b := 0.444444444 10^{-7}
> c:=a+b;
                          c := 1.00000004$
> c/b;
                         0.225000009 10^{8}
```

На этот раз получили ответ 22500000.9.

Рассмотрим еще пример.

Пример 2. Вычислим значения выражений: $\sin(x+y)-\sin(x)$ и $2\sin(y/2)\cos(x+y/2)$ при $x = 1$, $y = 1.0\cdot 10^{-7}$. Очевидно существует цепочка тригонометрических преобразований, переводящая первое выражение во второе. Имеем при вычислениях в среде MAPLE:

```
> restart;
> Digits:=8: x:=1;
                              x := 1
> y:=1.0*10^(-7);
                              y := 0.10000000  10^{-6}
> sin(x+y)-evalf(sin(x));
                             0.6  10^{-7}
> 2*sin(y/2)*cos(x+y/2);
                             0.54030232  10^{-7}
```

Результаты, как видим, разные, хотя аналитические выражения тождественны. Первый результат менее точен из-за вычитания достаточно близких чисел.

Снова обратимся к вычислениям в среде MAPLE.

Пример 3. Вычислим $(a-b)+c$ и $(a+c)-b$ при $a = 237$, $b = 236$, $c = 10^{-4}$.

```
> Digits:=5:
> a:=237: b:=236: c:=evalf(10^(-4)):
> f:=(a-b);
> R1:=f+c;
                             R1 := 1.0001
> f1:=(a+c);
```

```
                                f1 := 237.0
> R2:=f1-b;
                                R2 := 1.0
```

Если при точном выполнении операции сложения, вычитания, умножения и деления обладают свойствами ассоциативности, коммутативности и дистрибутивности, то эти свойства при выполнении тех же операций с округлением пропадают. Это доставляет много хлопот при конструировании численных методов.

Пример 4. Вычислим теперь $\sqrt{a} - \sqrt{b}$ и

$$\frac{a-b}{\sqrt{a}+\sqrt{b}} \text{ при } Digits = 8,\ a = 10000001,\ b = 10000000.$$

Очевидно, что эти выражения тождественны. Имеем.

```
> Digits:=8:a:=evalf(10000001);b:=evalf(10000000);
                  a := 0.10000001  10^{8}
                  b := 0.10000000  10^{8}
> g1:=sqrt(a)-sqrt(b);
                  g1 := 0.0001
> g3:=(a-b)/(sqrt(a)+sqrt(b));
                  g3 := .00015811388
```

Ответы опять получили разные.

Пример 5. Рассмотрим выражение

$$Z(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!} =$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots = sin(x).$$

Как известно $Z(\pi) = \sin(\pi) = 0$.

При вычислении в среде MAPLE при $x = 3.141$, $x_1 = 3.142$ получаем

```
> restart;
> Digits:=3;
> x:=3.141; z:=x-x^3/3!+x^5/5!-x^7/7!;
                        z := -.0744
>x1:=3.142; z1:=x1-x1^3/3!+x1^5/5!-x1^7/7!+x1^9/9!;
                        z1 := .00643
> z1-z;
                         .08083
```

Получили $z(\pi) \in [-.0744, .00643]$

Пример 6.

Вычислим в среде MAPLE при $Digits = 10$ скалярное произведение

$$(A, y) = \sum_{i=1}^{n} A_i y_i,$$

где

$$A = (10^{20}, 1223, 10^{18}, 10^{15}, 3, -10^{12},$$
$$y = 10^{20}, 2, -10^{22}, 10^{13}, 2111, 10^{16}.$$

```
 > restart;A:=[10^20,1223,10^18,10^15,3,-10^12];
> y:=[10^20, 2,  -10^22,  10^13,  2111,   10^16];
> #вычисляем скалярное произведение
> A[1]*y[1]+A[2]*y[2]+A[3]*y[3]+A[4]*y[4]+A[5]*y[5]+A[6]*y[6];
                                  8779
```

Итак,

$$(A, y) = \sum_{i=1}^{n} A_i y_i = 8779.$$

Вычислим еще раз при другом значении параметра $Digits$, возьмем $Digits$.

```
> Digits:=18;
> AF:=evalf(A);  yf:=evalf(y);
> AF[1]*yf[1]+AF[2]*yf[2]+AF[3]*yf[3]+AF[4]*yf[4]+AF[5]*yf[5]+AF[6]*yf[6];

                                  0.
```

Как видим, теперь получили в ответе 0.

Пример 7. Вычислим $x_{11} - x_{21}$ при разном количестве значащих цифр в мантиссе, $x_{11} = 0.1000548241 \cdot 10^5$, $x_2 1 := 0.9997342213 \cdot 10^4$. Имеем

```
>  x11:=0.1000548241*10^5;x21:=0.9997342213*10^4;
                        x11 := 10005.4824100000
                        x21 := 9997.3422130000
> x11-x21;
                              8.1401970000
> Digits:=5;
                              Digits := 5
> x11:=evalf(0.1000548241*10^5);x21:=evalf(0.9997342213*10^4);
                             x11 := 10005.
                             x21 := 9997.3
> x11-x21;
                              7.7
```

Отсюда видно, что при десяти значащих цифрах в мантиссе получаем $x_{11} - x_{21} = 8.1401970000$, а при пяти значащих цифрах в мантиссе получаем $x_{11} - x_{21} = 7.7$

1. Замер времени. Работа с системными таймерами на языке Си

В OpenMP предусмотрены функции для работы с системным таймером. Функция `omp_get_wtime()` возвращает в вызвавшем потоке астрономическое время в секундах (вещественное число двойной точности), прошедшее с некоторого момента в прошлом.

Если некоторый участок программы окружить вызовами данной функции, то разность возвращаемых значений покажет время работы выбранного участка.

Гарантируется, что момент времени, используемый в качестве точки отсчета, не будет изменён за время существования процесса.

Таймеры разных потоков могут быть не синхронизированы и выдавать различные значения.

Функция `omp_get_wtick()` возвращает в вызвавшем потоке разрешение таймера в секундах. Это время можно рассматривать как меру точности таймера.

Рассмотрим применение функций `omp_get_wtime()` и `omp_get_wtick()` для работы с таймерами в OpenMP.

В приведенном ниже фрагменте производится замер начального времени, затем происходит замер конечного времени.

Разность этих значений даёт замер времени выполняемого фрагмента программы.

Здесь также измеряется точность системного таймера.

```
#include "stdafx.h"
#include <stdio.h>
#include <omp.h>
int _tmain(int argc, char *argv[])
{
    double s_t, e_t, tick;
    s_t =omp_get_wtime();
    ..................... // фрагмент программы
    .....................
    e_t = omp_get_wtime();
    tick =omp_get_wtick();
```

```
    printf(" Замер времени в секундах %lf\n", e_t-s_t);
    printf("Точность таймера %lf\n", tick);
     return 0;
}
```

Большинство конструкций в OpenMP — это директивы компилятору вида `#pragma omp construct[clause [clause]...]`.

Для удобства дальнейшего изложения, перечислим основные функции OpenMP (версии 2.0).

1. `omp_set_num_threads` — устанавливает количество параллельных потоков.

2. `omp_get_num_threads` — выдает количество потоков в параллельной области.

3. `omp_get_max_threads` — выдает максимально возможное количество потоков в параллельной области.

4. `omp_get_thread_num` — выдает номер потока в параллельной области.

5. `omp_get_num_proc` — выдает число вычислительных элементов (процессоров или ядер), доступных приложению.

6. `omp_set_dynamic` — устанавливает режим динамического создания потоков.

7. `omp_get_dynamic` — выдает режим динамического создания потоков.

8. `omp_set_nested` — устанавливает режим вложенных параллельных областей.

9. `omp_get_nested` — выдает режим поддержки вложенных параллельных областей.

В OpenMP имеется два типа блокировок: простые и вложенные. Вложенные имеют суффикс «nest». Блокировки могут находиться в одном из трех состояний — неинициализированном, заблокированном и разблокированном.

10. `omp_init_lock`

`omp_init_nest_lock` — инициализирует замок (инициализация переменной типа `omp_lock_t/omp_nest_lock_t`).

11. `omp_set_lock`, `omp_set_nest_lock` — устанавливает замок (один поток выставляет блокировку, а остальные потоки ждут, пока

поток, вызвавшая эту функцию, не снимет блокировку с помощью функции `omp_unset_lock()`).

12. `omp_unset_lock`, `omp_unset_nest_lock` — снимает замок.

13. `omp_test_lock`, `omp_test_nest_lock` — устанавливает замок без блокировки.

14. `omp_destroy_lock`, `omp_destroy_nest_lock` — освобождение переменной типа `omp_lock_t/omp_nest_lock_t` .

15. `omp_in_parallel` — осуществляет проверку, является ли фрагмент параллельной областью.

16. `omp_get_wtime`, `omp_get_wtick` — функции замера времени.

Простые блокировки (`omp_lock_t`) не могут быть установлены более одного раза, даже тем же потоком. Вложенные блокировки (`omp_nest_lock_t`) идентичны простым с тем исключением, что когда поток пытается установить уже принадлежащую ему вложенную блокировку, он не блокируется.

Теперь перечислим основные директивы OpenMP.

Директивы OpenMP

1. `parallel` — устанавливает параллельную область.

Используемые параметры: `if`, `private`, `shared`, `default`, `firstprivate`, `reduction`, `copyin`, `num_threads`.

2. `for` — предназначается для распараллеливания циклов.

Используемые параметры: `private`, `firstprivate`, `lastprivate`, `reduction`, `ordered`, `nowate`, `schedule`.

3. `parallel for` — устанавливает параллельную область для распараллеливания цикла.

4. `sections` — предназначается для выделения области программного кода, в котором будут располагаться секции.

5. `section` — предназначается для образования секции в области программного кода `sections`.

6. `parallel sections` — предназначается для выделения параллельной области программного кода, в котором будут располагаться секции.

7. `master` — предназначается для выделения программного кода в параллельной области, который будет исполняться только потоком-мастером.

8. `single` — предназначается для выделения программного ко-

да в параллельной области, который будет исполняться только одним потоком.

9. `ordered` — предназначается для управления порядком вычислений в распараллеливаемом цикле.

10. `critical` — предназначается для образования критических секций.

11. `barrier` — предназначается для барьерной синхронизации потоков.

12. `atomic` — предназначается для атомарной (неделимой) операции.

13. `flush` — предназначается для синхронизации состояния памяти.

14. `threadprivate` — предназначается для определения в потоке локальных переменных.

Технология OpenMP часто называют технологией FORK/JOIN (вилка-объединение). Сначала присутствует последовательная часть программы, которая выполняется одним потоком, он называется потоком-мастером. Далее происходит распараллеливание выполнения программы по потокам, а затем вычисления проводятся одним потоком-мастером.

Первый закон Амдала

Производительность вычислительной системы, состоящей из связанных между собой устройств определяется самым непроизводительным устройством.

Предположим, что всего в алгоритме N операций и n операций из N приходится вычислять последовательно.

Отношение $f = n/N$ назовем долей последовательных вычислений.

Итак, доля операций, которые нужно выполнять последовательно — f, где $0 \leq f \leq 1$ (при этом доля понимается не по статическому числу строк кода, а по числу операций в процессе выполнения).

При этом возможны крайние случаи:
полностью параллельные ($f = 0$),
полностью последовательные ($f = 1$) программы.

Второй закон Амдала

Ускорение S которое может быть получено на компьютере из 'p' процессоров при данном значении f:

$$S \leq \frac{1}{f + (1-f)/p}$$

где p — число процессоров.

Следует отметить следующее. Если $9/10$ программы исполняется параллельно, а $1/10$ последовательно, то ускорения более, чем в 10 раз получить в принципе невозможно вне зависимости от качества реализации параллельной части кода и числа используемых процессоров (10 получается только когда время исполнения параллельной части равно 0).

Запуск последовательной программы.

Упражнение 1. Напишем последовательную программу, которая печатает слова "hello world".

Приведем фрагмент возможного решения.

```
void _tmain()
{
  int ID = 0;
 printf(" Hello(%d) ", ID);
 printf(" world(%d) \n", ID);
}
```

Запуск первой параллельной программы.

Большинство конструкций в OpenMP — это директивы компилятору вида

`#pragma omp construct[clause [clause]...]`.

Для того, чтобы фрагмент выполнялся в параллельном режиме, используем директиву

`#pragma omp parallel`.

В начале программы подключаем библиотеку `#include "omp.h"`.

Сколько физических процессоров существует на данном компьютере легко увидеть, запустив следующую программу.

Не забудем включить поддержку OpenMP в диалоговом окне свойств проекта ("Configuration Properties | C/C++ | Language OpenMP support -> YES).

Упражнение 2.

```
#include "omp.h"
#include <stdio.h>
int _tmain(int argc, char *argv[])
{
   printf("Последовательная область 1\n");
   #pragma omp parallel
   {
      printf("Параллельная область\n");
   }
   printf("Последовательная область 2\n");
 }
```

Можете ли прокомментировать полученный результат?

Ответ. В начале печатается "Последовательная область 1 затем по директиве parallel порождаются новые потоки, каждый из которых напечатает текст "Параллельная область затем порождённые потоки объединяются и оставшийся поток-мастер напечатает текст "Последовательная область 2". Количество этих потоков обычно совпадает с количеством имеющихся на компьютере физических потоков (не занятых в это время).

Можно распараллеливать выполнение программы на достаточно произвольном числе потоков, которое задаем специальной командой. Например, если нужно, чтобы вычисления проводились на 16 процессорах, можно применить директиву

`#pragma omp parallel num_threads(16)`.

```
#include "omp.h"
#include <stdio.h>
#define NUM_THREADS 16
```

```
int _tmain(int argc, char *argv[])
{
   printf("Последовательная область 1\n");
   omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel
   {
      printf("Параллельная область\n");
   }
   printf("Последовательная область 2\n");
 }
```

Упражнение 3.

Напишем программу, которая печатает слова "hello "world"и выведем номера потоков, на которые распределяются слова "hello "world"

```
#include "stdafx.h"
#include "omp.h"  // подключение OpenMP
#include <stdio.h>
#define NUM_THREADS 16
 void _tmain()
  {
  omp_set_num_threads(NUM_THREADS);
 #pragma omp parallel // начало параллельной области
   {
   int ID = omp_get_thread_num();
   printf(" hello(%d) ", ID);
   printf(" world(%d) \n", ID);
    } // конец параллельной области
  }
```

Данную программу следует запустить на выполнение несколько раз подряд. Консольное окно является общим ресурсом и какой поток достигнет его первым заранее неизвестно, поэтому при каждом новом запуске может получится новая комбинация слов и номеров соответствующих потоков.

О распараллеливании циклов

Неотъемлемой частью практически любой программы является цикл. Для распараллеливания должно быть указано конкретное количество витков цикла.

Распараллеливать цикл не всегда возможно. Например, если для выполнения итерации i нужно знать результат итерации $i-1$, т. е. итерация i зависит от итерации $i-1$, то распараллелить невозможно. Например, при попытке распараллеливания цикла возникнет сообщение об ошибке

```
for(int i = 1; i <= n; ++i)
a[i] = a[i-1] ;
```

Итак, в циклах for, распределенных на несколько потоков, порядок выполнения итераций может быть существенным, если например, выполнение одной итерации как-либо зависит от результата выполнения предыдущих итераций.

Рассмотрим пример.

```
for(int i = 0; i < 10; i++)
   arr[i] = i;
#pragma omp parallel for
for (int i = 1; i < 10; i++)
    arr[i] = arr[i - 1];
for(int i = 0; i < 10; i++)
    printf("\narr[%d] = %d", i, arr[i]);
```

Теоретически эта программа должна вывести последовательность из нулей. Однако, на двухпроцессорной машине будет выведено некоторое количество нулей и некоторое количество произвольных цифр от 0 до 9 (это связано с тем, что итерации как правило делятся между потоками пополам).