В.Н.Говорухин, В.Г.Цибулин

ВВЕДЕНИЕ В MAPLE. МАТЕМАТИЧЕСКИЙ ПАКЕТ ДЛЯ ВСЕХ

Пакет аналитических вычислений Maple является мощным инструментом решения математических проблем. Более двух тысяч эффективно реализованных команд позволяют решать задачи алгебры, математического анализа, дифференциальных уравнений, статистики, теории графов и многие другие. Пакет включает развитую графическую библиотеку и язык программирования.

Данная книга поможет читателю познакомиться с пакетом Maple и начать работать с ним, решая возникающие исследовательские или учебные задачи. Освобождение от рутинных, чреватых ошибками выкладок позволит быстрее справиться с более сложными проблемами и сохранить время для созидательной работы. Возможность перехода от аналитических к прямым вычислениям и легкость визуализации получаемых результатов делают Maple естественной средой обитания для начинающих и профессионалов.

В книге описывается система Maple V Release 3 для IBM PC под управлением Windows. В то же время большая часть книги посвящена командам, языку и тем возможностям Maple, которые не зависят от типа используемой платформы (SUN, Mac и др.) и сохранились в новой версии -Release 4 Power Edition.

Книга предназначена для студентов и специалистов, изучающих и/или применяющих математику.

Оглавление

Введение	**5**
0.1. Первые команды	6
0.2. Структура книги	9
0.3. Благодарности	11
1. Среда Maple	**12**
1.1. Объекты	13
1.2. Типы переменных	14
1.3. Выражения	15
1.4. Команды Maple	15
1.5. Синтаксис	17
1.6. Стандартные функции	20
1.7. Справка	22
2. Аналитические преобразования	**24**
2.1. Операции с формулами	24
2.2. Преобразования типов	28
2.3. Операции оценивания	30
3. Элементарная математика	**34**
3.1 Операции с полиномами	34
3.2. Решение уравнений и неравенств	36
3.3. Геометрические пакеты	40
3.4. Планиметрия	41
3.5. Стереометрия	44

4. Математический анализ **47**
4.1. Пределы, суммы, ряды 47
4.2. Исследование функций 49
4.3. Дифференцирование и интегрирование 51
4.4. Разложение и приближение функций 55
5. Линейная алгебра **58**
5.1. Работа со структурой матрицы и вектора 60
5.2. Основные матричные и векторные операции 62
5.3. Решение задач линейной алгебры 63
5.4. Векторный анализ 67
6. Дифференциальные уравнения **69**
6.1. Точные и приближенные решения 69
6.2. Численные решения 72
6.3. Структура DESol 74
6.4. Пакет DEtools 76
7. Математические библиотеки **85**
7.1. Определение абстрактных операторов 85
7.2. Интегральные преобразования 87
7.3. Интерполяция 88
7.4. Ортогональные полиномы 89
7.5. Теория чисел 89
7.6. Статистика 91
7.7. Степенные разложения 93
7.8. Линейная оптимизация 94
7.9. Математическая логика 95
7.10. Теория графов 96
7.11. Аппроксимация функций 98
7.12. Комбинаторика 99
7.13. Группы и формы 101
7.14. Другие пакеты 101
8. Программирование **103**
8.1. Условный оператор 103
8.2. Операторы цикла 104
8.3. Процедуры-функции 106
8.4. Процедуры 107
8.5. Команды ввода/вывода 110
8.6. Создание собственных библиотек 113
8.7. Отладка программ 115
9. Графика в Maple **119**
9.1. Опции двумерной графики 120
9.2. Команды двумерной графики 123
9.3. Двумерные графические структуры 129
9.4. Опции трехмерной графики 131
9.5. Структуры трехмерной графики 132

9.6. Команды трехмерной графики 134
9.7. Иллюстративные графические команды 141
10. Мини-исследования **144**
10.1 Логистическое отображение 144
10.2.Разложение функции в ряд Фурье 153
10.3.Система Лоренца как маятник с обратной связью 155
10.4. Течение вязкой жидкости в канале 170
11. Заключение **178**
11.1. Maple на марше 178
11.2. Maple и университеты 180
11.3. Maple и периодика 180
11.4. Maple и Internet 180
11.5. Maple и другие пакеты 181
11.6. Книги по Maple 182
11.7. Maple V - The Power Edition 183
Список литературы **186**
12. Приложение 187
12.1. Пункты меню Windows-версии 187
12.2. Пункты меню двумерной графики 191
12.3. Пункты меню трехмерной графики 193
Алфавитный указатель **197**

Алфавитный указатель

! 15
" 18
% 19
' 19
* 15
+ 15
- 15
-> 106
/ 15
: 17
, 17
^ 15
_C 19
_N 19
_Z 19
` 19
A
abs 20
add 62, 93
addcol 62
addedge 96
addrow 62
addvertex 96
adj 63
adjoint 63
algebraic 154
ALIGNABOVE 130
ALIGNBELOW 130
ALIGNLEFT 130
ALIGNRIGHT 130
ambientlight 132
angle 45, 66
animate 128
animate3d 139
antisymmetric 58, 86
appendto 110
arccos 21
arccosh 21
arccot 21
arccoth 21
arccsc 21
arccsch 21
arcscs 21
arcsech 21

arcsin 21
arcsinh 21
arctan 21
arctanh 21
are_collinear 41, 44
are_concurrent 41, 44
are_harmonic 41
are_orthogonal 41
are_parallel 41, 44
are_perpendicular 41, 44
are_similar 41, 44
are_tangent 41, 44
area 42, 45
array 14, 32, 58
arrows 122
assign 18, 37
associative 86
assume 47
asympt 56
axes 121, 131
B
basis 66
bequal. 95
Bessell 21
BesselJ 7, 21
BesselK 21
BesselY 21
Beta 21
bezout 68
binary 86
binomial 99
bisector 42
blockmatrix 60
bsimp 95
by 104
C
C 112
canon 95
center 42, 45
centroid 42
cfracpol 90
changevar 54
charpoly 64
chebpade 97
chebyshev 56
choose 100
circle 41, 42
circumcircle 42
close 111
coeff 28, 35
coeffs 35
coeftayl 56
col 61
coldim 60
collect 25, 116
color 121, 132
colorstyle 128
colspace 66
combinat 100
combine 26
commutative 86
complex 39
compose 93
concat 61
concyclic 42
cond 63
confracform 97
conjugate 31
CONTOUR 131
contourplot 137
convert 7, 29, 30, 36, 52, 60
convexhull 42
coordinates 42, 45
coords 121, 125, 131
coplanar 45
copyinto 61
cos 20
cosh 21
cot 20
coth 21
crossprod 67
esc 20
csch 21
curl 67
CURVES 130
cylinderplot 137
D
D 52, 70

Dchangevar 77, 84
define 85, 86
definite 65
degree 35
delcols 61
delrows 61
denom 25, 26
densityplot 126, 173
DEplot 76, 77
DEplotl 77, 81
DEplot2 77, 81
describe 91
DESol 69, 74
Del 63
del 63
detailf 42
determine 101
DEtools 69, 73, 76, 77, 157
dfieldplot 77, 83
diagonal 58
diameter 42
DiflF 51
diff 51, 70, 156
difforms 101
Digits 13, 32, 53
Dirac 21, 87
discont 50
discrim 35
display 92, 128, 155, 169, 173
display 3d 139, 168
distance 42, 44, 45
distrib 95
distributions 92
diverge 67
divide 34, 35
do 104
dodecahedron 138
dotprod 67
Doubleint 54
dsolve 8, 69, 71, 159, 163, 164
dual 95
E
E 13
edges 97
Eigenvals 64
eigenvals 64
eigenvects 64
ellipse 43
else 103
end 107
enlif 104
environ 95
erf 21
ERROR 109, 153
eval 18, 31, 59
evalc 31
evalf 31
evalf(int) 53
evalhf 31
evalm 31, 62, 63
evalp 91
evalpow 93
example 23
exp 20
expand 24, 25, 26, 34
extend 61
extrema 50
F
factor 24, 26, 35, 145
false 13, 41
feasible 94
ffgausselim 65
FFT 87, 166
fi 103
fibonacci 100
fieldplot 122, 127
fieldplot3d 137
find_angle 43
fit 91
Float 74
font 122
for 104
forall 87
forget 101
fortvan 111
fourier 87
frac 31
frame 128, 139

from 104
fsolve 38, 148
G
G 89
GAMMA 21
gamma 13
Gauss 102
gausselim 65, 127
Gaussln 89
gaussjord 65
gcd 35
genfunc 102
genmatrix 68
geom3d 40, 44
geometry 40, 41
GF 102
GInearest 91
GIprime 91
global 108
grad 67
gradplot 122, 127
gradplot3d 137
GramSchmidt 66
GRID 132
grid 122, 126, 132
grobner 102
Group 12, 85
group 101
H
H 89
Heaviside 21, 87
heights 137
help 22
hermite 65
hexahedron 138
HIDDEN 131
hilbert 60
histogram -142
Horizontal 130
hornerform 98
htranspose 63
I
I 13
icosahedron 138
identity 58, 86
if 103
ifactor 6
iFFT 88
Im 31
implicitplot 122, 126, 173
implicitplot3d 137
importdata 91
in 105
incircle 43
indexed 14
infinity 13
infnorm 98, 99
inifunction 21
init 101
Input Region 12
instquence 12.8, 139, 169
Int 7, 52, 176
int 52, 171
Int [student] 54
inter 43, 45
interface 116, 120
interp 88
intersect 15
intpaits 54
inverse 63, 86, 93
inversion 43
invfourier 88
invlaplace 88
is_equilateral 42
is_right 42
iscont 50
isolate 26
isolve 39
isprime 90
J
jacobian 67
Jordan 65
K
kernel 65
L
L 89
labels 122, 126, 132
laplace 72

laplacian 67
lasterror 117
latex 112
laurent 56, 98
lcoeff 35
ldegree 35
leastsqrs 66
leastsquare 92
leftbox 141
LegendreE 21
LegendreEc 21
LegendreF 21
LegendreKc 21
LegendrePi 21
LegendrePic 21
length 32
lhs 26
liesymm 101
light 132
Limit 48
limit 48
linalg 16, 58, 64, 127
LINE 131
line 44
Iine3d 44
Linear 85
Lineint 54
linestyle 122
linsolve 66
list 32
listlist 132
In 20
local 108, 153
log 20
log 10 20
log[a] 20
logic 95
loglogplot 125
logp 91
logplot 125
lprint 111
M
makehelp 114
march 115
matrix 58
matrixplot 137
max 32, 33
maximize 50, 94
mellin 88
MESH 133
middlebox 141
midpoint 43, 45
min 32
minimax 98, 99
minimize 50, 94
minor 61
mint.exe 118
minus 15
msolve 39
mtaylor 56
mulcol 62
mulrow 62
multiply 62, 93
N
negative 93
Networks 96
nextprime 90
nops 27
norm 67
normal 25, 26
normalize 66
NPspinor 102
numapprox 56, 97
numbcomb 100
numbperm 100
numer 25, 26
numeric 72, 163
numpoints 121, 125, 132
numtheory 89
O
octahedron 138
od 104
odeplot 73, 76, 128, 137, 163
on_circle 42
on_line 42
on_plane 45
on_sphere 45, 46

op 27, 59
open 111
operator 108
options 108
Order 13, 32, 55, 56, 72
order 32
ordering 101
orhtopoly 89
orientation 131
orthog 66
orthopoly 8, 36
output 163
Output Region 12
P
P 89
pade 98
padic 89, 91
parallel 43, 45
PATCH 131
PATCHCONTOUR 131
PDEplot 77
permute 100
perpen_bisector 43
perpendicular 43, 45
phaseportrait 77, 83, 157
Pi 13
plane 44
PLOT 130, 168
plot 120, 123, 125
plot3d 134, 136, 137
plotdevice 116, 120
plotoutput 116, 120
plots 119, 163, 173
plottools 184
POINT 131
point 41, 44
point3d 44, 46
pointplot 137
POINTS 129
poisson 56
polar 31
polarplot 126
poligonplot 125
polygonplot3d 137
POLYGONS 130, 168
polyhedraplot 137
polyscale 137
polytype 138
powcreate 93
powdiff 93
powexp 93
powint 93
powlog 93
powpoly 93
powseries 93
powsolve 94
prettyprint 116
print 111
printf 111
printlevel 115
proc 107, 149, 150, 153, 169, 172
Product 49
product 49
projection 43, 45, 131, 138
projgeom 40
proot 36
psqrt 36
Q
quo 35
quotient 93
R
radius 43, 45, 46
rand 142
randbool 96
randmatrix 60, 127
random 91
randpoint 43
randpoly 36
rank 63
Re 31
read 110
readdata 113
readlib 15, 47, 112
readline 113
readshare 143
readstat 113
realroot 36
reflect 43, 45

related 23
rem 36
remember 108
remez 98
replot 128
residue 51
resolution 121
RETURN 109
reversion 93
rhs 8, 26, 147, 160
rightbox 141
RootOf 38, 71
round 31
row 61
rowdim 60
rowspace 66
rsolve 39
S
satisfy 96
save 110
scaling 121, 125, 131
scatter2d 92, 142
sec 20
sech 21
Separator 12
series 30, 32, 55, 71, 75
session 12
set 14, 32
setoptions 121
setoptions3d 132, 139
shading 132
share 143
showtangent 142
showtime 118
sides 43
signum 20
simplex 94
simplify 24, 25, 26, 28, 146
sin 20
singular 51
sinh 21
solve 6, 17, 20, 36, 66, 146
sort 32
spacecurve 138, 1
sphere 44, 45, 46
sphereplot 139
spline 88, 165
sqrt 20
stack 61
statevalf 91
statplots 91, 92
stats 16, 91, 113, 142
statsplot 142
string 13, 14, 154
student 53, 58, 141
style 121, 125, 131
submatrix 61
subs 7, 24, 26, 156, 157
subsop 27
subtract 93
subvector 61
Sum 48, 170
sum 19, 48
surfdata 139
swapcol 62
swaprow 62
Sylvester 68
symbol 122, 125
symmetric 43, 45, 58, 86
T
T 89
tan 20
tangent 43, 45
tangentpc 43
tanh 21
tassume 101
tautology 96
taylor 7, 55
terminal 110
tetrahedron 45, 138
TEXT 114, 130
Text Region 12
textplot 126, 154
textplot3d 139
thickness 121
tickmarks 132
time 118
tis 101

title 121, 131
titlefont 124
to 104
totorder 101
tpsform 93
trace 16, 63, 115
transform 91
transpose 63
traperror 117
triangle 44
triangle3d 45
trigsubs 26
Tripleint 54
true 13, 41
trunc 31
tubeplot 139
type 28, 154
U
U 89
unapply 99, 107, 145
unary 86
unassign 37
union 15
untrace 115
updates 22
usage 23
V
value 47, 48, 164
vectdim 60
vector 58
verboseproc 116
vertical 130
vertices 97
view 131
volume 45, 46
W
whattype 15, 28
while 104
WIREFRAME 131
with 16, 47, 114, 119
words 118
worksheet 6
write 111
writeln 111
writeto 110
X
xtickmarks 121
Y
ytickmarks 121
Z
zero 86
Zeta 21

Введение

В последние годы системы компьютерной алгебры (символьных или аналитических вычислений) стали доступны не только узкому кругу специалистов, но и всем обладателям персональных компьютеров. Сейчас несколько компаний предлагают мощные и развитые пакеты: Axyom, Derive, Macsyma, Maple, Mathematica, Reduce и др. Данная книга посвящена пакету Maple, который является одним из лидеров среди универсальных систем и обеспечивает пользователю удобную и интеллектуальную среду для математических исследований. Кроме того, символьный анализатор Maple включен в ряд пакетов вычислительного характера, таких, как MathCad и Matlab, и входит в состав пакетов подготовки научных публикаций (Scientific WorkPlace и Math Office).

Пакет Maple – совместное детище университета Ватерлоо (штат Онтарио, Канада) и Высшей технической школы (ETH, Цюрих, Швейцария). На английском языке вышло более 60 книг, описывающих Maple и его многочисленные применения в научных исследованиях и для преподавания разнообразных предметов. Сам пакет широко распространен в университетах ведущих научных держав, исследовательских центрах и компаниях. При этом пакет Maple развивается, вбирая в себя новые умения, новые разделы математики и обеспечивая лучшую среду для работы. Более подробно о развитии Maple, о его возможностях и перспективах см. в гл.11 "Заключение".

Maple состоит из ядра — процедур, написанных на языке C и в высшей степени оптимизированных, библиотеки, написанной на Maple-языке, и интерфейса. Ядро выполняет большинство базисных операций. Библиотека содержит множество команд – процедур, выполняемых в режиме интерпретации. Программируя собственные процедуры, пользователь может пополнять ими стандартный набор и, таким образом, расширять возможности Maple.

Интерфейс Maple в настоящее время зависит от используемой техники. В этой книге речь в основном пойдет о Maple-языке, который не меняется при переходе от машины к машине. Приводимые примеры были опробованы на IBM PC для Windows-версии, сведения об интерфейсе которой даны в Приложении. Этот интерфейс основан

на концепции рабочего поля (**worksheet**) или документа, содержащего строки ввода, вывода и текст, а также графику.

0.1. Первые команды

Предваряя более (или менее) регулярное изложение возможностей пакета Maple, попробуем на примере нескольких команд показать, что это простой, интуитивно понятный и удобный программный продукт.

Работа с пакетом происходит в режиме интерпретатора. В строке ввода пользователь задает команду, нажимает клавишу Enter и получает результат – строку (строки) вывода либо сообщение об ошибочно введенной команде. Тут же выдается приглашение вводить новую команду и т.д.

В этой книге примеры работы пакета выделяются чертой сбоку, а строки ввода команд отмечаются префиксом (>).

Поставляемый пакет Maple снабжен файлом quicktour.ms – поучительной и информативной экскурсией по возможностям пакета, которую мы рекомендуем совершить каждому. Начнем изложение со знакомства с кратким аналогом этого документа. Мы надеемся, что простота и интуитивная ясность большинства команд Maple и некоторые пояснения помогут понять содержание примеров.

Вычислим сумму квадратов целых чисел от 1 до 1996 и присвоим результат переменной S. Здесь и далее две точки подряд определяют диапазон изменения переменной.

```
> S:=sum(k^2,k=1..1996);
```

$$S := 2652690986$$

Разложим полученное число на простые множители.

```
> ifactor(S);
```

$$(2)(11)^3(499)(1997)$$

Решим систему двух линейных уравнений с неизвестными x, y и тремя параметрами a, b и c.

```
> z:=solve({a*x+b*y=1, -b*x+a*y=c}, {x,y});
```

$$z := \left\{x = -\frac{-a+bc}{a^2+b^2}, y = \frac{b+ca}{a^2+b^2}\right\}$$

Подставим в решение для x условие, что сумма квадратов a и b равна единице.

```
> subs(a^2+b^2=1, z[1]);
```

$$x = a - b\,c$$

Продифференцируем функцию Бесселя первого рода и результат присвоим переменной f.

```
> f:=diff(BesselJ(n,2*x),x);
```

$$f := -2\,\mathrm{BesselJ}(n+1, 2\,x) + \frac{n\,\mathrm{BesselJ}(n, 2\,x)}{x}$$

Дадим параметру n конкретное значение.

```
> n:=1;
```

$$n := 1$$

С учетом введенного n разложим выражение f, являющееся результатом дифференцирования функции Бесселя, в ряд Тейлора.

```
> p:=taylor(f,x);
```

$$p := 1 - \frac{3}{2}x^2 + \frac{5}{12}x^4 + O(x^5)$$

Преобразуем полученное разложение в полином.

```
> p:=convert(p,polynom);
```

$$p := 1 - \frac{3}{2}x^2 + \frac{5}{12}x^4$$

Вычислим определенный интеграл.

```
> Int(p,x=0..1)=int(p,x=0..1);
```

$$\int_0^1 1 - \frac{3}{2}x^2 + \frac{5}{12}x^4\,dx = \frac{7}{12}$$

Зададим линейное дифференциальное уравнение третьего порядка.

```
> de:=diff(y(x),x,x,x)-diff(y(x),x,x)+
>     diff(y(x),x)-y(x)=-exp(Pi)/2;
```

$$de := \left(\frac{\partial^3}{\partial x^3} y(x)\right) - \left(\frac{\partial^2}{\partial x^2} y(x)\right) + \left(\frac{\partial}{\partial x} y(x)\right) - y(x) = -\frac{1}{2} e^{\pi}$$

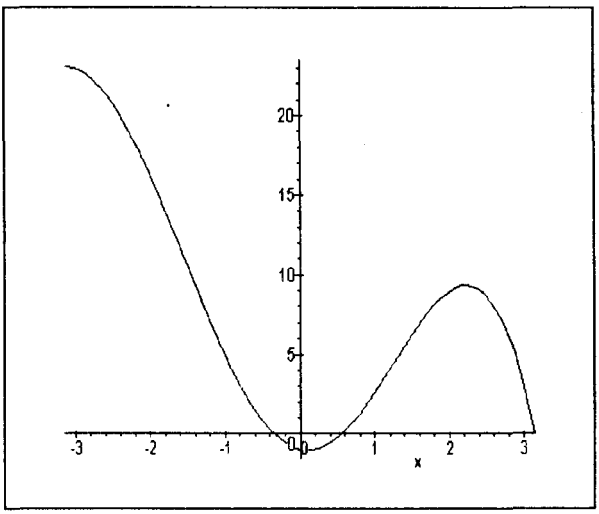
Найдем решение краевой задачи.

```
> sol:=dsolve({de,y(0)=-1,D(y)(0)=-1,y(Pi)=0}
>                  ,y(x));
```

$$sol := y(x) = \frac{1}{2} e^{\pi} - e^{x} - \frac{1}{2} e^{\pi} \cos(x)$$

Нарисуем график решения. Для этого при помощи команды **rhs** извлечем правую часть из результата предыдущей команды.

```
> plot(rhs(sol),x=-Pi..Pi);
```



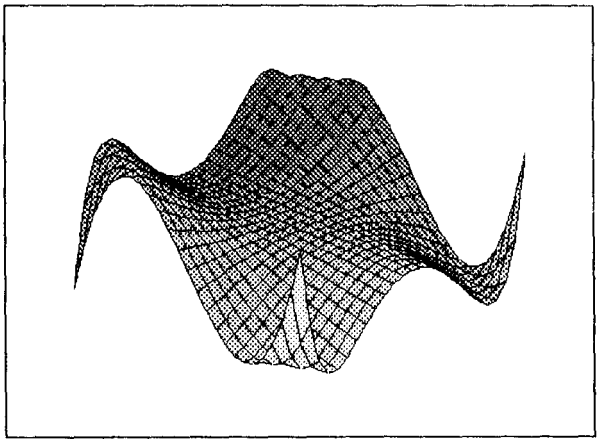
Подключим библиотеку ортогональных полиномов.

```
> with(orthopoly);
```

$$[G, H, L, P, T, U]$$

Нарисуем трехмерную поверхность, заданную при помощи полинома Чебышева первого рода.

```
> plot3d(T(3,x^2*y),x=-1..1,y=-1..1);
```



Приведенными примерами далеко не исчерпывается мощь пакета Maple; описание использованных в этих примерах и многих других команд будет дано ниже.

0.2. Структура книги

Книга состоит из предисловия, десяти глав, заключения, Приложения, списка литературы и алфавитного указателя.

В первой главе описана среда пакета: структура пакета, режим работы, объекты. Объяснены правила построения выражений и способы обращения к справочной системе. В конце приведен перечень обычных и специальных функций, известных Maple.

Вторая глава посвящена аналитическим преобразованиям формул и обсуждению обслуживающих эти преобразования команд. Здесь описаны команды преобразования выражений, подстановки, упрощения, перевода в другой формат представления данных.

Третья глава посвящена тому, что с некоторой натяжкой можно назвать элементарной математикой. Сюда включены операции с полиномами, материал о решении алгебраических уравнений и неравенств, а также сведения о геометрических пакетах Maple.

В четвертой главе изложены команды, реализующие содержание классического математического анализа: дифференцирование, интегрирование, вычисление пределов, суммирование, разложение в ряды и др.

В пятой главе описаны команды линейной алгебры и библиотека `linalg`. Это мощный инструмент для проведения всевозможных матричных операций, решения систем линейных уравнений, вычисления собственных значений и векторов.

В шестой главе представлены возможности Maple для аналитического, приближенного и численного решения систем дифференциальных уравнений. Рассмотрены способы задания уравнений, начальных и краевых условий, описана специальная конструкция для аналитической работы с решением.

Описание библиотек пакета Maple дано в седьмой главе. Множество команд Maple для решения задач комбинаторики, теории графов, математической логики, статистики и др. бегло изложено в этой главе.

В восьмой главе, посвященной программированию на языке Maple, перечислены основные конструкции циклов и условных операторов, описаны правила построения процедур и функций. Простота и гибкость языка Maple позволяют легко преодолевать трудности по автоматизации исследовательской работы.

Девятая глава посвящена графическим возможностям пакета. Разнообразие команд двумерной и трехмерной графики, а также желание осветить существенную их часть объясняют конспективность изложения.

В десятой главе приведены примеры мини-исследований – выполненных средствами Maple работ по изучению свойств логистического отображения, по исследованию системы дифференциальных уравнений, описывающей осциллятор с меняющейся формой потенциальной ямы, а также по применению метода Галеркина для расчета стационарного течения вязкой жидкости в канале квадратного сечения.

Заключение содержит разнообразные сведения: новости о последней реализации пакета, информацию о развитии программного обеспечения фирмы Maple Software, некоторые адреса, дающие доступ к Maple-кладовой Internet.

В Приложении описан интерфейс для Maple for Windows и его система меню.

Книга организована таким образом, чтобы читатель мог получить основные сведения о Maple, не отвлекаясь на подробное изучение каждой команды. Так как некоторые команды имеют широкий спектр возможностей и применяются во многих разделах Maple-

математики, то их описание дается по ходу изложения материала в разных главах. В конце книги дан алфавитный указатель всех описанных в книге команд.

0.3. Благодарности

Оглядываясь назад, авторы с удовольствием вспоминают, что эта книга не была бы написана без поддержки, прямой или косвенной, большого числа хороших людей. Спасибо им всем.

Мы признательны фирме Waterloo Maple Inc. и ее специалистам, снабжавшим нас ценной и своевременной информацией, а главное – создавшим и развивающим такой замечательный продукт [1,2].

Мы благодарны коллегам и товарищам по работе в Ростовском государственном университете за советы и ободрение.

Спасибо организаторам проекта EmNet/NIS Phase II, направленного на развитие математической инфраструктуры в государствах, вышедших из Советского Союза, и финансирующей это дело европейской организации INTAS.

Этой книге не суждено было бы выйти без помощи наших семей, и мы очень рады такой зависимости.

Приятно сознавать, что работа окончена, хотя, возможно, и не все описки и огрехи устранены. Авторы будут благодарны всем замечаниям читателей, а также предложениям и комментариям. Наш адрес для переписки:

maple-gts@mmf.unird.ac.ru

1. Среда Maple

Начнем с технических требований. В данной книге описывается система Maple V Release 3, работающая на IBM PC под управлением Windows 3.1. Это связано с распространенностью IBM-совместимых машин в России и укоренившимся стандартом на использование графических оболочек типа Windows. В то же время большая часть книги посвящена командам, языку и тем возможностям Maple, которые не зависят от типа используемой платформы (SUN, Mac и др.).

Для установки пакета нужен компьютер с процессором от 80386, с 13 мегабайтами на жестком диске и минимум с 4 мегабайтами оперативной памяти. Наличие лучшей машины, больших тактовой частоты и оперативной памяти повышает комфортность работы и дает возможность решать более сложные задачи.

Работа в Maple проходит в режиме сессии **(session)** — пользователь вводит предложения (команды, выражения, процедуры), которые воспринимаются и интерпретируются Maple. То, что при этом появляется на экране дисплея, условно разделяется на три части: область ввода (**Input Region**), которая состоит из набираемых предложений, область вывода (**Output Region**) и тексты комментариев (**Text Region**). Область вывода может включать результаты выполнения математических и алгоритмических операций, а также графические образы (двумерная и трехмерная графика). Область ввода и вывода вместе с сопутствующими комментариями называется группой (**Group**). Группы разделяются сепараторами (**Separator**).

Нажатие клавиши Enter запускает исполнение предложения. Если введено законченное предложение, то следует выполнение, иначе — Maple ожидает его завершения. Обнаружив ошибку, Maple печатает на следующей строке сообщение о ней; при синтаксической ошибке символом "^" отмечается первая неузнанная литера.

Результаты работы могут быть сохранены в файлах различных форматов. Имеются два вида Maple-файлов. По умолчанию все результаты работы (области ввода и вывода, комментарии) записываются в файл с расширением 'ms'. Если задан режим сохранения состояния сессии (активен режим 'Save Kernel'), то, кроме того, в файле

с расширением 'm' будут записаны текущие назначения переменных и коды введенных процедур. При записи в файлы с другими расширениями сохраняются только области ввода и тексты комментариев.

1.1. Объекты

Простейшими объектами в Maple являются числа, константы, строки и имена. Числа могут быть целыми, рациональными, корнями и числами с плавающей запятой, например:

$$589,\ 12/7,\ \sqrt{3},\ 3.34562134568906\,E0$$

Операции с рациональными числами и корнями позволяют проводить абсолютно точные вычисления, так как отсутствует погрешность округления. Операции с вещественными числами проводятся по умолчанию с десятью значащими цифрами, но, переопределив зарезервированную константу **Digits**, можно работать с любой мантиссой. Это может быть полезно и при символьных вычислениях, поскольку операции с рациональными числами выполняются медленнее. Даже в студенческой версии (Student Version) доступны операции с числами до 100 знаков.

В Maple представлены все основные математические константы. Перечислим важнейшие из них:

Pi — число π,
E — e (основание натурального логарифма),
I — мнимая единица,
infinity — бесконечность,
gamma — константа Эйлера,
true, false — булевские величины.

Имена этих констант являются зарезервированными, а их значения не могут быть переопределены, в отличие от ряда управляющих констант (**Digits, Order**).

Строкой (**string**) является любой набор символов, заключенный в обратные кавычки. Например:

```
`This is a Maple string`
```

Каждая переменная Maple имеет имя – набор символов, начинающийся с буквы, причем большие и малые буквы

различаются. Кроме букв могут употребляться цифры и знак подчеркивания. Приведем примеры различных имен:

```
NewValue, newvalue, new_value, n1
```

В качестве имен переменных запрещено использовать слова Maple-языка:

and, by, do, done, elif, else, end, fi, for, from, if, in, intersect, local, minus, mod, not, od, option, options, or, proc, quit, read, save, stop, then, to, union, while.

Кроме того, ряд имен команд также запрещено использовать в качестве имен.

1.2. Типы переменных

В Maple существует множество типов переменных: от известных вещественного (**float**), целого (**integer**) и массива (**array**) до тех, которые необходимы в аналитических преобразованиях: дробь (**fraction**), функция (**function**), индексная переменная (**indexed**), процедура (**procedure**), строка (**string**), множество (**set**), разложение (**series**), последовательность выражений (**exprseq**) и некоторые другие. Например, важным элементом Maple являются последовательности или списки (переменные типа **list**), идеально подходящие для хранения коэффициентов полиномов, асимптотических разложений и т.д.

По умолчанию переменная считается скалярной и имеющей тип **string**. Это фактически математическая переменная, как x в формуле $f(x)$. Для задания переменных других типов требуется явное их определение: при помощи оператора присваивания или команд, преобразующих тип. Например, для заведения массива из трех элементов с именем **a** используется следующая конструкция:

```
> a:=array(1..3);
```

$$a:=array(1..3,[\,])$$

Здесь и далее в тексте знаком "больше" (**>**), как уже говорилось, отмечено приглашение ввода.

Сама переменная **a** при этом считается строковой (**string**), а любой элемент массива – индексной переменной (**indexed**). Если

ввести через запятые несколько величин и взять их в фигурные скобки, то получится переменная типа **set** (множество). Именно таким образом предстают перед пользователем найденные Maple корни уравнения и задаются системы уравнений.

Информацию о типе той или иной переменной можно получить при помощи команды **whattype**.

1.3. Выражения

Используя переменные и знаки арифметических и других операций, можно составлять выражения. Знаками операций являются:

+ — сложение,
- — вычитание,
* — умножение,
/ — деление,
^ — возведение в степень,
! — факториал.

Последовательность выполнения арифметических операций соответствует стандартным математическим правилам: сначала проводится возведение в степень, затем умножение и деление, а в конце – сложение и вычитание. Операции выполняются слева направо, для изменения порядка используются круглые скобки.

В булевых операциях также применяются знаки **>**, **<**, **>=**, **<=**, **<>**, **=**.

Для работы со множествами имеются специальные операции **intersect** (пересечение), **minus** (разность), **union** (объединение).

1.4. Команды Maple

Выражения и переменные обычно служат параметрами команд Maple. Стандартная команда выглядит следующим образом:

```
command(par1,par2,...,parn);
```

Здесь **command** – имя команды, а **par1,par2,...,parn** – ее параметры.

Часть команд Maple вызывается автоматически, перед использованием других необходимо загрузить их в память командой **readlib**. Кроме того, многие команды являются частью пакетов

(библиотек), и до запуска команды пакет должен быть загружен командой

```
with(package).
```

Здесь **package** – имя пакета. Такими пакетами являются: **DEtools**, **Gausslnt**, **genfunc**, **geometry**, **geom3d**, **group**, **liesymm**, **logic**, **networks**, **NPspinor**, **numapprox**, **padic**, **projgeom**, **totorder**.

Если нужен вызов единственной команды **command** из пакета **package**, то можно пользоваться следующими конструкциями:

```
with(package,command),
```

или

```
package[command](par1,par2,...,parn),
```

где **package** – имя пакета, **par1**, **par2**,..., **parn** – параметры команды.

Заметим, что в последних реализациях появились подпакеты, способы обращения с которыми можно узнать, например обратившись к документации по пакету **stats**.

Некоторые команды переопределяются при подключении ряда пакетов, так, например, команда трассировки **trace** после подключения пакета линейной алгебры **linalg** замещается командой вычисления следа матрицы. Предупреждение о переопределенных командах выдается при загрузке пакета.

Предупредим об опасности произвольного назначения имен переменных. Если имя переменной совпадает с именем какой-нибудь команды, то такая команда становится недоступной в текущем сеансе. Поэтому перед введением новой переменной **name** полезно удостовериться, что имя не занято, командой

```
?name
```

Появление окошка справки будет означать существование команды с именем **name**.

Если команда введена правильно и полностью, то Maple выполняет ее и приводит в следующих строках результат. Если появилось эхо введенной команды или область вывода пуста, то либо Maple отказывается выполнить команду из-за неполноты информации, либо не может ее исполнить (уравнение не решается, интеграл не

берется и т.п.). В этом случае полезно задуматься о том, что делается: подключена ли нужная библиотека, решается ли в принципе поставленная задача, нет ли других подходов, методов, команд.

Наш совет начинающему пользователю: внимательно читайте результат выполнения команды. Maple – очень гибкая система, старающаяся выполнить любую введенную команду; так что даже если вы, например, не подгрузили библиотеку, то какой-то результат все равно будет. Всегда проверяйте полученное размышлением, ведь использование системы аналитических вычислений не отменяет мыслительного процесса, а только помогает ему. Например, если в строке ввода опустить знак умножения перед круглой скобкой в выражении *x*(y-1)*, то Maple посчитает, что *y-1* является аргументом некоторой функции *x(y-1)*.

1.5. Синтаксис

Каждая команда должна завершаться разделителем: точкой с запятой (;) или двоеточием (:). Если ввод предложения завершается разделителем (;), то в строке под предложением сразу будет отклик: результат исполнения команды или сообщение об ошибке. Разделитель (:) используется для отмены вывода, когда команда выполняется системой, но ее результат не выводится на экран.

В Maple применяются круглые, квадратные и фигурные скобки. Назначение круглых скобок – задавать порядок при построении математических выражений и обрамлять аргументы функций и параметры в записи команд. Квадратные скобки нужны для работы с индексными величинами. Фигурные скобки используются для формирования множеств, например системы уравнений.

Для построения Maple-предложений нужны: знак равенства (=) при формировании уравнений и знак присвоения (:=) при задании значений переменных. Различие в их использовании можно проиллюстрировать следующим примером решения системы двух уравнений. Аргументами команды решения уравнений **solve** являются множество уравнений (заключенные в фигурные скобки и разделенные запятой два уравнения) и множество переменных, для которых ищется решение. Результат решения присваивается некоторой переменной **sols**.

```
> sols:=solve({x+y=3,2*x=y},{x,y});
```

В результате получается множество решений

$$sols := \{x = 1, y = 2\}$$

Как видно, получаемые решения представляют собой набор уравнений для указанных переменных. Если решений несколько, то выдаются все найденные решения (заметим, что это не обязательно все решения). Переменные x и y, однако, остаются неназначенными (неопределенными). Для того чтобы их определить, нужно воспользоваться командой **assign**:

```
> assign(sols); x;
```

$$1$$

В этом примере строка ввода состоит из двух команд, причем вторая выдает значение переменной x.

Вообще, для выдачи значения любой скалярной переменной достаточно в строке ввода указать имя самой переменной и завершить команду разделителем (;). Для просмотра содержимого индексных переменных (например, с именем **arr**) используется команда

```
eval(arr).
```

Приведем пример задания двумерного массива и выдачи его содержимого. Заметим, что на введенное имя массива последовало только его эхо.

```
> A:=array(1..2,1..2,[[a,b],[c,d]]): A;
```

$$A$$

```
> eval(A);
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Здесь для указания размерности массива использован знак диапазона – две идущие подряд точки. Вообще, в Maple две последовательные точки в опциях команд применяются для определения интервала изменения переменных.

Теперь расскажем о трех видах кавычек, встречающихся в Maple. Назначение двойных кавычек – экономить усилия при обращении к полученным результатам вычислений. Просто двойными кавычками (") обозначается предшествующий вывод. Пара двойных

кавычек отсылает к предпоследнему результату. Наконец, предшественник предпоследнего результата обозначается тремя кавычками. Эта нотация может быть удобна при последовательной работе с документом, но чревата неприятностями при свободном перемещении по тексту, когда команды выполняются в произвольном порядке.

Обратные кавычки (`) указывают на строку символов, причем две обратные кавычки подряд позволяют включить саму кавычку в строку.

Прямые кавычки (') (апостроф) используются для того, чтобы освободить переменную от предшествующих назначений. Например:

```
> expr:=x^2+exp(y): expr;
```

$$x^2 + e^y$$

```
> expr:='expr': expr;
```

$$expr$$

Кроме того, прямые кавычки полезны для предупреждения ошибки в том случае, когда для выполнения команды используется переменная, получившая значение ранее. Например, последовательность команд

```
> i:=3;
```

$$i:=3$$

```
> sum(i^2,i=1..6);
```

приведет к ошибке, поскольку Maple воспримет второй аргумент в команде суммирования **sum** как 3=1..6. Применение кавычек даст правильный результат:

```
> sum('i^2','i'=1..6);
```

$$91$$

При работе с пакетом следует иметь в виду, что иногда в ответе **(Output Region)** Maple использует специальные обозначения. Для указания неопределенных констант в решении дифференциального уравнения применяются символы *_C1*, *_C2* и т.д. Целая константа обозначается как *_N1*, *_N2*, ..., а комплексная величина соответственно как *_Z1*, *_Z2* и т.д. При выводе выражения для сокращения записи также вводятся переменные с именами *%1*, *%2* и т.д. Эти переменные можно использовать независимо от самого выражения. Следующий пример показывает, как возникает вспомогательная переменная *%1* при представлении корня кубического уравнения. Здесь

указание 2 в квадратных скобках говорит о том, что требуется выдать второй корень (результатом команды **solve** в общем случае является массив решений).

```
> solve(x^3-x-a,x)[2];
```

$$-\frac{1}{2}\%1^{1/3}-\frac{1}{6}\frac{1}{\%1^{1/3}}+\frac{1}{2}I\sqrt{3}\left(\%1^{1/3}-\frac{1}{3}\frac{1}{\%1^{1/3}}\right)$$

$$\%1:=\frac{1}{2}a+\frac{1}{18}\sqrt{-12+81\,a^2}$$

1.6. Стандартные функции

Обращение к стандартной функции – элементарная математическая операция, если известно имя нужной функции. Список изначально определенных в Maple функций велик, поэтому просто перечислим достаточно представительное подмножество из известных функций. Начнем с таблицы, устанавливающей соответствие между несколькими математическими функциями и соответствующими именами функций Maple:

Математическая запись	Maple-запись
e^x	`exp(x)`
$\ln x$	`ln(x)` или `log(x)`
$\log_{10} x$	`log10(x)`
$\log_a x$	`log[a](x)`
$\sqrt{x}$	`sqrt(x)`
$\lvert x \rvert$	`abs(x)`
$\operatorname{sgn} x$	`signum(x)`
$n!$	`n!`

Далее перечислим несколько семейств функций, для которых достаточно указать имя.

Тригонометрические функции (аргументы в радианах):

```
sin, cos, tan, cot, sec, csc.
```

Обратные тригонометрические функции:

arcsin, arccos, arctan, arccot, arcscs, arccsc.

Гиперболические функции:

sinh, cosh, tanh, coth, sech, csch.

Обратные гиперболические функции:

arcsinh, arccosh, arctanh, arccoth, arcsech, arccsch.

Дельта-функция Дирака и функция Хевисайда:

$\delta(t)$ - **Dirac(t)**,

$H(t)$ - **Heaviside(t)**.

Бесселевы функции:

$J_p(x)$ - **BesselJ(p,x)**,

$Y_p(x)$ - **BesselY(p,x)**,

$I_p(x)$ - **BesselI(p,x)**,

$K_p(x)$ - **BesselK(p,x)**.

Гамма- и бета-функции:

GAMMA(z) и **GAMMA(a,x)**,
Beta(x,y).

Дзета-функция Римана:

Zeta(z).

Интеграл ошибок:

erf(x).

Эллиптические интегралы в форме Лежандра первого, второго и третьего рода:

LegendreF, LegendreE, LegendrePi.

Полные эллиптические интегралы в форме Лежандра первого, второго и третьего рода:

LegendreKc, LegendreEc, LegendrePic.

Справку о всех имеющихся в Maple функциях можно получить, выполнив команду

?inifunction.

1.7. Справка

Хорошая справочная система Maple доступна как через пункты меню (см. Приложение), так и непосредственно из командной строки. Для этого удобно использовать специальный формат справки – строку из вопросительного знака и имени команды, но без разделителя в конце. Например, запрос о пакете, команде или служебном слове **topic** выглядит следующим образом:

```
?topic
```

Заметьте, что эта команда не завершается разделителем. Через запятую или в квадратных скобках можно указать слово (подтему) из некоторого общего раздела (темы). Если запрашивается информация о команде **subtopic** из пакета **topic**, то соответствующий запрос оформляется при помощи команд

```
?topic,subtopic
```

```
?topic[subtopic]
```

К аналогичным справкам приведет использование команды **help** с обычным разделителем (;) в конце:

```
help(topic);
```

```
help(topic,subtopic);
```

```
help(topic[subtopic]);
```

Если при наборе имени команды, о которой запрашивается информация, была сделана ошибка, то Maple выдаст перечень имен, которые могут иметь отношение к запросу. Это окажется полезным, если вы не помните полностью имени команды, а также при интуитивном способе освоения Maple, когда пользователь предполагает, что для некоторой операции может существовать команда с таким-то именем.

Кстати, начиная работу с Maple, полезно справиться о последних новостях реализации, которые могли быть не отмечены в руководстве. Здесь поможет запрос

```
?updates
```

Как всякое чтение расширяет кругозор, так и данная справка может сообщить много интересного из того, чем не станешь интересоваться специально. Именно после прочтения последних новостей авторы узнали о следующих двух запросах, предназначенных для вывода информации о формате команды **topic**

```
??topic
```

и

```
usage(topic).
```

Окошко справки с примерами для команды **topic** появится после ввода конструкции

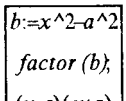
```
???topic
```

или

```
example(topic);
```

Наконец, сведения о родственных для **topic** командах возникнут в области вывода после запроса

```
related(topic).
```



2. Аналитические преобразования

Хотя некоторые сведения о командах Maple появились в предыдущих главах, именно с данной главы начинается изложение того разнообразия возможностей системы Maple, которое заключено в ее командах и пакетах. Наиболее часто используемые основные математические и общесистемные команды находятся в стандартной библиотеке, которая автоматически подключается после запуска Maple.

Следующие главы посвящены описанию команд, выполняющих основные математические операции интегрирования, дифференцирования, разложения в ряды, суммирования, решения уравнений и неравенств и т.д., здесь же остановимся на командах общего назначения. Начнем с описания ряда команд для преобразования выражений (упрощения, подстановки, приведения к нужному виду и др.), а также для изменения их типов.

2.1. Операции с формулами

При преобразовании математических выражений обычно приходится делать ряд рутинных операций: приводить подобные члены, раскладывать на множители (факторизовать), раскрывать скобки, делать подстановки и др. Maple обладает широкими возможностями для проведения выкладок — аналитических преобразований математических формул. Имена команд, реализующих эти операции, соответствуют английским терминам и просты: **factor** – для факторизации, **expand** – для раскрытия скобок, **simplify** – для упрощения выражения, **subs** – для подстановки и т.д.

Начнем с простых примеров, иллюстрирующих применение некоторых команд для работы с алгебраическими выражениями. Будем предварять примеры самих команды и результаты их работы короткими пояснениями.

Присвоим переменной *w* дробь, у которой числитель и знаменатель являются алгебраическими выражениями от *x* и *y*. Для разложения на множители этого выражения применим команду **factor**.

```
> w:=(x^2+y^2)/(x^4-y^4); z:=factor(w);
```

$$w := \frac{x^2+y^2}{x^4-y^4}$$

$$z := \frac{1}{(x-y)(x+y)}$$

Видно, что помимо факторизации выполнена и нормализация (сокращение одинаковых множителей дроби). Просто для сокращения (нормализации) дроби имеется команда **normal**, а команды **numer** и **denom** позволяют выделить соответственно числитель и знаменатель дроби.

```
> normal(w); numer(z); d:=denom(z);
```

$$\frac{1}{x^2-y^2}$$

$$1$$

$$d := (x-y)(x+y)$$

Далее раскроем скобки в выражении с именем *d* при помощи команды **expand** и проведем подстановку, заменив *x* и *y* тригонометрическими функциями аргумента *t*.

```
> expand(d);u:=subs({x=cos(t),y=sin(t)},d);
```

$$x^2-y^2$$

$$u := (\cos(t)-\sin(t))(\cos(t)+\sin(t))$$

Наконец, при помощи команды **simplify** упростим последнее полученное выражение

```
> simplify(u);
```

$$2\cos(t)^2-1$$

Приведенные примеры показывают естественность принятого в Maple формата команд и простоту проведения выкладок. Назначение ряда других команд можно понять из нижеприводимого перечня команд, данного в алфавитном порядке.

collect(expr,var) — приведение подобных членов в выражении **expr** относительно переменной **var**.

combine(expr) — приведение подобных членов в выражении **expr**, содержащем известные функции, например, sin, exp.

denom(rat) — выделение знаменателя рациональной дроби **rat**.

expand(expr) — раскрытие скобок.

factor(expr) — разложение выражения на множители.

isolate(eqn,expr) — определение выражения **expr** из уравнения **eqn**. Перед использованием команда должна быть подгружена командой **readlib(isolate)**.

lhs(eqn) — выделение левой части из уравнения **eqn**.

normal(rat) — нормализация дроби – сокращение общих множителей и приведение к виду числитель/знаменатель.

numer(rat) — выделение числителя рациональной дроби **rat**.

rhs(expr) — выделение правой части уравнения **eqn**.

simplify(expr,<opt1,...,optn>) — упрощение выражения **expr** согласно заданным опциям (**@**, **Ei**, **exp**, **GAMMA**, **hypergeom**, **ln**, **polar**, **power**, **radical**, **RootOf**, **sqrt**, **trig**). Например, при указании опции **trig** упрощение производится с использованием большего числа тригонометрических соотношений. В качестве опций можно задавать также соотношения в виде равенств, тогда упрощение будет проводиться с учетом этих соотношений.

subs(old=new, expr) — подстановка выражения **new** вместо **old** в выражение **expr**.

trigsubs(expr) — выдача всех тригонометрических эквивалентов выражения **expr**. Перед использованием команда должна быть подгружена командой **readlib**.

Заметим, что простые подстановки можно делать при помощи оператора присваивания, когда переменной присваивается некоторое выражение. Однако в этом случае введенная подстановка распространится и на следующие операции с участием этой переменной.

Проиллюстрируем действие ряда команд на операциях с тригонометрическими функциями.

```
> eq:=sin(4*x)+sin(2*x)+sin(x)=cos(2*x);
```

$$eq := \sin(4x) + \sin(2x) + \sin(x) = \cos(2x)$$

```
> trigsubs(rhs(eq)); factor(simplify(lhs(eq),trig));
```

$$\left[\cos(2x), \cos(2x), 2\cos(x)^2 - 1, 1 - 2\sin(x)^2, \cos(x)^2 - \sin(x)^2, \frac{1}{\sec(2x)}, \frac{1}{\sec(2x)}, \frac{1-\tan(x)^2}{1+\tan(x)^2}, \frac{1}{2}e^{(2Ix)} + \frac{1}{2}e^{(-2Ix)}\right]$$

```
> readlib(isolate): isolate(eq,sin(x));
```

$$\sin(x) = \cos(2x) - \sin(4x) - \sin(2x)$$

Для квалифицированной работы с формулами требуются некоторые знания о структурах Maple. Во внутреннем представлении Maple-объект делится на подобъекты и т.д., вплоть до базисных элементов, так что получается ветвящаяся, древоподобная структура. При работе с большими выражениями часто требуется извлекать отдельные элементы структуры и преобразовывать их. Команда **nops** сообщает о количестве объектов первого уровня, а команда **op** выдает их в виде последовательности выражений.

Проиллюстрируем сказанное примером:

```
> object:=x^2+2*x-3;
```

$$object := x^2 + 2x - 3$$

```
> nops(object);
```

$$3$$

```
> op(object);
```

$$x^2, 2x, -3$$

Кроме того, при помощи команды **op** можно извлекать подвыражения, указывая номер объекта первого уровня. Например,

```
> op(1,object);
```

$$x^2$$

Для подстановки выражения **new** в n-й операнд первого уровня выражения **expr** используется команда

subsop(n=new,expr).

Может получиться, что Maple выдаст выражение не в той форме, которую вы ожидали или предпочитаете. Чтобы упростить выражение, перейти от экспоненциальных функций к тригонометрическим и т.п., может применяться команда **convert**, которая

будет описана ниже. Кроме перечисленных имеются и другие команды преобразования формул, например позволяющие работать с внутренней структурой выражения. Информацию о назначении и форматах команд можно получить, обратившись к Справке Maple.

2.2. Преобразования типов

Каждая команда Maple работает только с определенными типами данных, поэтому если тип фактического параметра не отвечает допустимому для данной команды, то будет выдано сообщение об ошибке. В этом случае перед использованием команды следует преобразовать выражение, приведя его к нужному типу.

Остановимся на двух важных командах, помогающих разобраться с типами выражений. Для элементарных действий эти команды не очень нужны, но обязательны для квалифицированного использования Maple.

type(expr,kind) — выяснение вопроса, является ли выражение **expr** объектом типа **kind**, где **kind** может быть одним из следующих терминов: **const**, **form**, **laurent**, **scalar**, **series**, **taylor**.

Полный список терминов можно получить, обратившись к Справке, а для *извлечения точного значения любого термина* **kind** следует запросить справочную систему, набрав команду

```
?type[kind].
```

whattype(expr) — определение типа выражения **expr**. Результатом будет выдача одного из терминов: *****, **+**, **.**, **..**, **<**, **<=**, **<>**, **=**, **^**, **and**, **array**, **exprseq**, **float**, **fraction**, **function**, **indexed**, **integer**, **list**, **not**, **or**, **procedure**, **series**, **set**, **string**, **table**, **uneval**.

Ниже приведен пример, в котором команда **whattype** помогает выяснить причину ошибки при применении команды **coeff** для определения коэффициента при квадрате переменной x. После действия команды **simplify** получено выражение типа произведения, поэтому необходимо собрать коэффициенты при

одинаковых степенях x, преобразовав выражение в полином, чтобы затем определить нужное.

```
> f:=simplify(x^2/a+x-1);
```

$$f := \frac{x^2 + x\,a - a}{a}$$

```
> coeff(f,x^2);
```

Error, unable to compute coeff

```
> whattype(f);
```

*

```
> g:=collect(f,x);
```

$$g := \frac{x^2}{a} + x - 1$$

```
> whattype(g);
```

+

```
> coeff(g,x^2);
```

$$\frac{1}{a}$$

Теперь перейдем к команде **convert**, назначение которой преобразовывать одни объекты в другие с изменением их типа.

convert(expr,option) — преобразование выражения **expr** со-гласно заданной опции.

В качестве опции **option** могут выступать следующие термины: **`+`**, **`*`**, **D**, **base**, **binary**, **binomial**, **confrac**, **decimal**, **degrees**, **diff**, **double**, **equality**, **exp**, **expln**, **expsincos**, **factorial**, **float**, **fraction**, **GAMMA**, **hex**, **horner**, **hostfile**, **hypergeom**, **lessthan**, **lessequal**, **list**, **listlist**, **ln**, **matrix**, **metric**, **mod2**, **multiset**, **name**, **octal**, **parfrac**, **polar**, **polynom**, **radians**, **radical**, **rational**, **ratpoly**, **RootOf**, **set**, **sincos**, **sqrfree**, **tan**, **trig**, **vector**.

Поясним некоторые случаи употребления команды.

convert(list,array) — преобразование списка **list** в одномерный массив **array** с теми же элементами.

convert(list,vector) — преобразование списка **list** в вектор **vector** с теми же элементами.

`convert(list1,...,listn,matrix)` — преобразование списков `list1,...,listn` в матрицу `matrix`.
`convert(s,poly)` — преобразование списка `s` в полином `poly`.
`convert(s,ratpoly)` — преобразование списка `s` в полиномиальное выражение с рациональными коэффициентами.
`convert(poly,list)` — преобразование полинома `poly` в список `list`.

Мнемоника помогает понять назначение части терминов: (`binary`, `float`, `radians`), о некоторых разновидностях команды `convert` будет рассказано ниже, а за получением точного значения любого термина `termin` следует обратиться к справочной системе, набрав команду

```
?convert[termin].
```

Следующий пример показывает как, вычислив разложение функции `cos` в ряд Тейлора, перевести его в полином и найти четвертый член ряда, воспользовавшись для этого командой `convert`

```
> s:= series(cos(x),x,7);
```

$$s := 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 + O(x^7)$$

```
> p:=convert(s,polynom);
```

$$p := 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6$$

```
> q:=convert(p,list);
```

$$q := \left[1, -\frac{1}{2}x^2, \frac{1}{24}x^4, -\frac{1}{720}x^6\right]$$

```
> q[4];
```

$$-\frac{1}{720}x^6$$

2.3. Операции оценивания

Целое семейство команд посвящено операциям оценивания. Только для переменной скалярного типа ее значение будет выдано, если в строке ввода указать имя переменной. Для переменных сложных типов аналогичная строка ввода вызовет область вывода,

состоящую только из имени переменной. Чтобы посмотреть содержимое таких переменных, нужна команда **eval**.

eval(array) — выдача содержимого массива **array**.

Ряд команд служит для вычисления выражения с преобразованием к нужному типу. Если в выражении **expr** все числа заданы при помощи рациональных выражений (дроби, степени), то для перехода к числам с плавающей запятой применяется команда

evalf(expr)

Кроме того, для ускорения вычислений с плавающей запятой имеется команда **evalhf**, использующая арифметику компьютера. По умолчанию все операции в Maple производятся символьно.

Для вычисления значения комплексного выражения **cmplx** используется команда

evalc(cmplx)

Для вычисления матричного выражения **expr** с матрицами в качестве операндов и допустимыми операторами (&*), (+), (–), (^) имеется команда

evalm(expr).

Для операций с вещественными числами необходимы следующие команды:

frac(expr) — вычисление дробной части действительного выражения **expr**.
round(expr) — округление действительного выражения **expr**.
trunc(expr) — вычисление целой части выражения **expr**.

Для работы с комплексными числами полезно помнить о командах:

conjugate(cmplx) — вычисление комплексно-сопряженной величины для комплексного выражения **cmplx**.
Im(cmplx) — определение мнимой части выражения **cmplx**.
polar(cmplx) — преобразование комплексного выражения **cmplx** в полярные координаты.
Re(cmplx) — определение действительной части выражения **cmplx**.

Приведем пример, использующий некоторые из этих команд.

```
> a:=2/3*(x+I*y)^2+sin(Pi*sqrt(3)); evalf(a);
```

$$a := \frac{2}{3}(x+Iy)^2 + \sin(\pi\sqrt{3})$$

$$.6666666667\,(x+1.\,Iy)^2 - .7458348278$$

```
> Re(a); evalc(Re(a));
```

$$\frac{2}{3}\Re((x+Iy)^2) + \sin(\pi\sqrt{3})$$

$$\frac{2}{3}x^2 - \frac{2}{3}y^2 + \sin(\pi\sqrt{3})$$

Несколько простых команд общего назначения не требуют особого комментария.

`array(1..n,list)` — создание вектора (одномерного массива) из `n` элементов списка `list`.

`length(str)` — вычисление длины строковой переменной `str`.

`max(exp1,..expn)` — нахождение максимума из выражений `exp1,...,expn`.

`min(exp1,..expn)` — нахождение минимума из выражений `exp1,...,expn`.

`sort(list)` — сортировка последовательности `list`.

Следующие три команды понадобятся для проведения всевозможных разложений.

`order(s)` — вывод текущего значения порядка усечения для последовательности `s` (переменная типа `series`, например результат разложения выражения в ряд Тейлора).

`Order:=n` — определение значения глобальной переменной `Order`.

`Digits:=n` — определение длины мантиссы для операций с плавающей запятой.

Приведем простой пример, использующий некоторые из описанных команд и демонстрирующий использование фигурных и квадратных скобок для превращения последовательности выражений (здесь это числа 1,9,9,6) в переменные типа `set` (множество) и `list` (список).

```
> year:=1,9,9,6; whattype(year);
```

$$year := 1, 9, 9, 6$$

exprseq

```
> whattype([year]); whattype({year});
```

list

set

```
> max(year); a:=sort([year]);
```

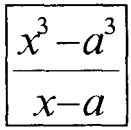
9

$$a := [1, 6, 9, 9]$$

```
> convert(a,set); {year};
```

$$\{1, 6, 9\}$$

$$\{1, 6, 9\}$$



3. Элементарная математика

Определение "элементарная" в названии этой главы относится к фундаментальному для математики набору команд, входящих в ядро Maple – стандартную библиотеку. Здесь описаны операции с полиномами и команды решения алгебраических уравнений и неравенств. Во второй части главы описаны геометрические пакеты Maple.

3.1. Операции с полиномами

Для всех пакетов аналитических вычислений операции с полиномами являются базовыми и часто используются при других преобразованиях формул. Под полиномом в Maple понимается сумма выражений с неотрицательными степенями, так что полиномами являются константа, простая переменная и выражение. Полиномы бывают одной или нескольких переменных.

Ниже дано представительное подмножество команд работы с полиномами, которое предваряется достаточно простым примером. Введем полином **p** двух переменных **x** и **y**, умножим его на выражение **x+1**, используем команду **expand** для раскрытия скобок и результат присвоим переменной **q**. Затем при помощи команды **divide** разделим получившееся выражение на полином **y+1**. Далее выведем полученный в результате деления полином **p1**, определим его степень (команда **degree**) и коэффициент при старшей степени (команда **lcoeff**). Наконец, при помощи команды **gcd** найдем наибольший общий делитель полинома **p1** и полинома $x^2 - 2x$.

```
> p:=x*y+x-2*y-2;
```

$$p := x\,y + x - 2\,y - 2$$

```
> q:=expand(p*(x+1));
```

$$q := x^2\,y - x\,y + x^2 - x - 2\,y - 2$$

```
> divide(q,y+1,p1);
```

true

```
> p1; degree(p1);
```

$$-2 + x^2 - x$$

```
> lcoeff(p1);
```

$$2$$

```
> gcd(p1,x^2-2*x);
```

$$1$$

$$x-2$$

Теперь приведем список команд с короткими пояснениями.

coeff(poly,var,int) – вычисление коэффициента при n-ой степени переменной **var** для полинома **poly**.

coeffs(poly,var) – вычисление коэффициентов полинома **poly** при переменной (переменных) **var**. Результатом будет переменная типа **list**, содержащая неупорядоченные коэффициенты. Для получения соответствия между коэффициентами и степенями переменной **var** нужно использовать расширенный формат команды с указанием третьего параметра, см. пример.

```
> coeffs(x^2+y*x-x^4+y^2,x,`s`); s;
```

$$y^2, 1, y, -1$$

$$1, x^2, x, x^4$$

degree(poly,var) – вычисление степени полинома **poly** по переменной **var**.

discrim(poly,var) – вычисление дискриминанта полинома **poly** по переменной **var**.

divide(poly1,poly2,name) – вычисление частного от деления двух полиномов **poly1** и **poly2** присваивает переменной **name**. Результат **true** при успешном делении и **false** – при неудаче.

factor(poly) – разложение полинома на множители.

gcd(poly1,poly2) – вычисление наибольшего общего делителя двух полиномов.

lcoeff(poly,options) – вычисление старшего коэффициента полинома.

ldegree(poly,var) – вычисление наименьшей степени полинома относительно переменной **var**.

quo(poly1,poly2,var) – вычисление частного от деления двух полиномов.

rem(poly1,poly2,var) – вычисление остатка от деления двух полиномов.

Команда **convert** для операций с полиномами имеет несколько специфических видов:

convert(poly,horner,var) – преобразование полинома по схеме Горнера.
convert(poly,mathorner,var) – преобразование полинома по матричной форме схемы Горнера.
convert(poly,sqrfree,var) – разложение полинома **poly** на квадратные трехчлены.

Перед запуском двух следующих команд **proot** и **psqrt** их нужно подгрузить из стандартной библиотеки при помощи команд **readlib**.

proot(poly,n) – нахождение корня **n**-ой степени из полинома.
psqrt(poly) – нахождение квадратного корня из полинома.

В заключение приведем еще две полезные команды:

randpoly(var) – создание случайного полинома переменной **var**.
realroot(poly) – вычисление интервала, где лежат вещественные корни полинома **poly**.

Для операций с полиномами могут быть полезны также команды из пакета работы с ортогональными полиномами **orthopoly**, некоторые возможности которого описаны в гл. 7 "Математические библиотеки".

3.2. Решение уравнений и неравенств

Для аналитического решения алгебраических уравнений используется команда

solve(eqns,vars)

Здесь **eqns** – уравнение или система уравнений, а **vars** – переменная или группа переменных в фигурных скобках. Система уравнений задается в виде множества. Напомним, что множеством

является совокупность разделенных запятыми объектов, взятая в фигурные скобки.

Уравнения могут быть заданы непосредственно в теле команды, а могут быть введены приравниванием выражений некоторой переменной, например **eqns**. Если в качестве первого параметра команды **solve** ввести выражение, то приравнивание этого выражения нулю производится автоматически.

Для хранения решений удобно ввести переменную и обращаться к конкретному решению по индексу, например:

```
> sols:=solve(x^3+x=0,x);
```

$$sols := 0, I, -I$$

```
> sols[2];
```

$$I$$

Отметим, что результатом решения одного уравнения будет переменная типа **exprseq**, а если в качестве параметра команды **solve** взять уравнение в фигурных скобках, то результатом будет переменная типа **set**.

```
sols:=solve({x^3+x=0},x);
```

$$sols := \{x = 0\}, \{x = I\}, \{x = -I\}$$

Для того чтобы присвоить найденные значения переменным, относительно которых искалось решение, применяется команда **assign**. Эта команда эквивалентна операции присваивания. Для отмены назначения переменной применяется команда **unassign** или эквивалентная операция присваивания переменной ее имени, взятого в кавычки. Нижеследующий пример демонстрирует это.

```
> s:=solve({a*x-y=sqrt(3), 5*x+a*y=1},{x,y});
```

$$s := \left\{y = -\frac{-a + 5\sqrt{3}}{a^2 + 5}, x = \frac{a\sqrt{3} + 1}{a^2 + 5}\right\}$$

```
> assign(s); x; y;
```

$$\frac{a\sqrt{3} + 1}{a^2 + 5}$$

$$-\frac{-a + 5\sqrt{3}}{a^2 + 5}$$

```
> unassign('x'); y:='y': x; y;
```

$$x$$

$$y$$

При достаточном количестве времени и памяти Maple всегда отыщет решение системы линейных уравнений. Для нелинейных уравнений может быть найдено несколько решений (но не обязательно все), а может оказаться, что решение не найдено. Тогда Maple просто выдаст приглашение ввода, ожидая новой команды. Если не указаны переменные, относительно которых должно быть найдено решение, то ответ будет выдан для всех переменных.

```
> solve(a*x+b=0);
```

$$\{b = -a\,x, x = x, a = a\}$$

Если в выражении ответа появилась функция **RootOf**, это означает, что Maple не может выразить корни в радикалах, и решение выражается через корни аргумента этой функции.

Конструкция **RootOf(a(x)=0,x)** используется для обозначения любого корня уравнения **a(x)=0**. Расширенный формат команды позволяет указать тот корень из множества корней, который находится вблизи значения **z**

RootOf(a(x)=x,x,z).

При этом можно выделять также комплексные корни, задавая комплексные значения для **z**. Например,

```
> y1:=RootOf(x^4-16=0,x,3); evalf(y1);
```

$$y1 := \mathrm{RootOf}(\_Z^4 - 16, 3)$$

$$2.$$

```
> y2:=RootOf(x^4-16=0,x,-1.7*I); evalf(y2);
```

$$y2 := \mathrm{RootOf}(\_Z^4 - 16, -1.7\,I)$$

$$-2.000000000\,I$$

Для численного решения системы уравнений **eqns** относительно переменных **vars** используется команда **fsolve**

fsolve(eqns,vars,option).

Здесь при помощи параметров **option** могут быть заданы дополнительные условия, устанавливающие, что

complex – разыскиваются комплексные корни;
a..b – для поиска корней задан интервал **[a,b]**;
maxsols=n – определено число разыскиваемых решений;
fulldigits – используется арифметика с максимальной мантиссой.

По умолчанию Maple старается найти все вещественные корни.

```
> fsolve(x^4-x^3-x^2-x-2,x);
                          -1., 2.
> fsolve(cos(x)=x);
                        .7390851332
```

Помимо описанных команд **solve** и **fsolve** имеется ряд специализированных команд: **isolve(eqn)** – для отыскания решений уравнения **eqn** в целых числах, **msolve(eqn,m)** – для нахождения решений уравнения **eqn** по модулю **m**.

Для нахождения решений систем линейных разностных уравнений с постоянными коэффициентами и некоторых нелинейных разностных уравнений (рекуррентных соотношений) имеется команда

rsolve(eqns,fncs).

Здесь **fcns** – имя функции (набор имен функций), относительно которой (которых) будет решаться разностное уравнение (система уравнений) **eqns**. Если решение может быть получено, то будет выдан ответ в виде функции от параметра. Помимо самих уравнений в **eqns** также могут содержаться начальные или граничные условия. При их отсутствии Maple постарается выдать ответ в общем виде. Следующий пример показывает, как получается решение линейного разностного уравнения второго порядка.

```
> eq:=2*f(n)=3*f(n-1)-f(n-2):
> rsolve({eq,f(1)=0,f(2)=1},f);
```

$$2-4\left(\frac{1}{2}\right)^n$$

Команда **solve** применяется также для решения неравенств. Например,

```
> solve(2*ln(x^2-3)<3-ln(x^2-3)^2,x);
```

$$\left\{x < -\sqrt{e^{(-3)}+3}, -\sqrt{e+3} < x\right\},$$
$$\left\{\sqrt{e^{(-3)}+3} < x, x < \sqrt{e+3}\right\}$$

Для решения задач с несколькими неравенствами, например для задач линейного программирования, имеется пакет **simplex**.

3.3. Геометрические пакеты

У Maple есть три геометрических пакета: **geometry** – для задач планиметрии, **geom3d** – для задач стереометрии и **projgeom** – для проективной геометрии. Перед обращением к командам пакета сам пакет должен быть подгружен при помощи команды **with**. В каждом пакете кроме команд задания геометрических объектов (линии, плоскости, окружности, сферы и т.д.) имеются команды для определения некоторых характерных величин (площадь, объем и др.), а также ряда менее стандартных величин (например линии Эйлера). Все геометрические структуры, определенные при помощи одного из этих пакетов, могут использоваться только в пределах действия данного пакета. Нельзя, например, в команде пакета **geom3d** ссылаться на точку, заданную при помощи команды из пакета **geometry**.

Для всех геометрических пакетов характерен следующий способ определения объектов: первый параметр команды задает имя объекта и далее следует собственно информация об объекте. Этим геометрические пакеты отличаются от обычного в Maple определения какой-нибудь переменной или структуры при помощи операции присваивания. Здесь также действует правило оценки последнего имени, как и для всех таблично задаваемых структур. Например, для просмотра полей структуры, описывающей точку, нужно использовать команду **op**.

```
> with(geometry):
> point(p,[1,2]);
> p;
```

$$p;$$

```
> op(p);
   table([
     x=1
```

```
        form=point
        y=2
        ])
> pt[x];
                              1
```

Далее коротко охарактеризуем команды геометрических пакетов. Конспективность изложения объясняется намерением компании, производящей Maple, переписать эти пакеты в новых реализациях.

3.4. Планиметрия

Пакет **geometry** содержит команды для решения задач двумерной евклидовой геометрии. Перед началом работы пакет нужно подгрузить. Латинские буквы **x** и **y** используются как глобальные переменные для координат точек, а также в качестве переменных в уравнениях прямых и окружностей. Геометрические объекты определяются обычным образом: точка задается своими координатами (команда **point**), прямая – двумя точками или уравнением (команда **line**), окружность (**circle**) – тремя точками, уравнением, заданием центра и радиуса, диаметром.

Целый ряд команд пакета проверяет выполнение того или иного условия для геометрических объектов. Мнемоника здесь вполне понятная, и при возможности определенного ответа результатом является булевская константа (**true** или **false**); в некоторых случаях выдаются координаты объекта (например, точки), при которых будет выполнено проверяемое условие. Приведем ряд команд, опуская параметры, если их количество и назначение следуют из пояснения.

are_collinear – проверка, лежат ли три точки на одной прямой.

are_concurrent – проверка, проходят ли три прямые через одну точку.

are_harmonic – проверка двух точек на гармоническую сопряженность двум другим точкам.

are_orthogonal – проверка двух окружностей на ортогональность.

are_parallel – проверка параллельности двух прямых.

are_perpendicular – проверка перпендикулярности двух прямых.

are_similar – проверка двух треугольников на подобие.

are_tangent(line,circle) – проверка, касательна ли прямая **line** окружности **circle**.

concyclic – проверка существования окружности, которой принадлежат четыре точки.

is_equilateral – проверка треугольника на равносторонность.

is_right – проверка треугольника на прямоугольность.

on_circle(pt,circle) – проверка принадлежности точки **pt** окружности **circle**.

on_line(pt,line) – проверка принадлежности точки **pt** прямой **line**.

Для многих команд результат действия присваивается переменной с именем **name**. Задание большинства геометрических объектов (вершины треугольника, точки отрезка и др.) не должно содержать никаких символических переменных.

area – вычисление площади заданного объекта (треугольника, круга или квадрата).

bisector(tri,pt,name) – вычисление отрезка от вершины **pt** до середины противоположной стороны треугольника **tri**. У этой команды имеется синоним – **median**.

center(circle) – определение центра окружности, результат присваивается переменной **center_circle**.

centroid(tri,name) – вычисление центра тяжести треугольника.

circle(name,[pt,expr]) – вычисление окружности с центром в точке **pt** и радиусом **expr**.

circle(name,[pt1,pt2,pt3]) – вычисление окружности, проходящей через три точки.

circumcircle(name,tri) – вычисление описанной вокруг треугольника **tri** окружности.

convexhull – вычисление окружности, проходящей через три точки из заданного множества точек так, что все остальные точки содержатся внутри окружности.

coordinates(pt) – вывод координат точки **pt**.

detailf(arg) – вывод информации об аргументе **arg**, в качестве которого может быть точка, прямая или окружность.

diameter – вычисление диаметра круга, содержащего заданные точки.

distance(pt,line) – вычисление расстояния между точкой **pt** и прямой **line**.

ellipse(name,[]) – определение эллипса одним из следующих способов: по пяти точкам, по центру и двум полуосям или при помощи уравнения.

find_angle – вычисление угла между двумя прямыми или двумя окружностями.

incircle(tri,name) – вычисление вписанной в треугольник **tri** окружности.

inter(line1,line2) – вычисление точки пересечения двух прямых.

inversion(pt,circle,name) – вычисление для точки **pt** точки инверсии относительно окружности **circle**.

midpoint(pt1,pt2,name) – вычисление средней точки на отрезке, заданном двумя точками **pt1** и **pt2**.

parallel(pt,line,name) – вычисление прямой, проходящей через точку **pt** и параллельной прямой **line**.

perpen_bisector(pt1,pt2,name) – вычисление прямой, проходящей черезе середину отрезка, заданного двумя точками **pt1** и **pt2**, и ортогональной ему.

perpendicular(pt,line,name) – вычисление прямой, проходящей через точку **pt** и перпендикулярной прямой **line**.

projection(pt,line,name) – вычисление проекции точки **pt** на прямую **line**.

radius – вычисление радиуса окружности.

randpoint(line,name) – задание случайной точки на прямой **line**.

reflect(pt,line,name) – вычисление точки, зеркально симметричной точке **pt** относительно прямой **line**.

sides – вычисление периметра треугольника.

symmetric(pt,line,name) – вычисление точки, которая симметрична точке **pt** по отношению к прямой **line**.

tangent(pt,circle,name1,name2) – вычисление двух прямых, проходящих через точку **pt** и касательных к окружности **circle**; результат присваивается переменным **name1** и **name2**.

tangentpc(pt,circle,name1) – вычисление касательной к окружности **circle**, проходящей через точку **pt**.

Поясним действие ряда команд примером. После подключения геометрического пакета зададим три точки A, B, F, затем определим проходящие через них прямые AB, AF, а прямую CD введем с помощью уравнения. Далее проверим параллельность прямых и вычислим расстояние от точки F до прямой CD. В заключение определим треугольник и найдем его площадь.

```
> with(geometry):
> point(A,[0,1]); point(B,[1,0]): point(F,[1,1]):
                                  A
> line(AB,[A,B]); line(AF,[A,F]): line(CD,[x+y=2]):
                                  AB
> are_parallel(AB,CD);
                                 true
> are_parallel(AB,AF);
                                 false
> distance(F,CD);
                                  0
> triangle(ABS,[A,B,point(S,[b,a])]): area(ABS);
                              -1/2+1/2 a+1/2 b
```

3.5. Стереометрия

Команды пакета трехмерной геометрии **geom3d** похожи на рассмотренные команды двумерной геометрии.

Для определения точки, прямой, плоскости и сферы применяются соответственно функции **point3d**, **line3d**, **plane** и **sphere**. Для пакета **geom3d** идентификаторы **x**, **y**, **z** и **_t** используются глобально для указания координат точек, а также в уравнениях прямых, плоскостей и сфер.

Аналогично командам двумерной геометрии ряд команд выполняет проверку некоторых геометрических условий.

are_collinear – проверка, лежат ли три точки на одной прямой.

are_concurrent – проверка, проходят ли три прямые через одну точку.

are_parallel – проверка параллельности двух прямых.

are_perpendicular – проверка перпендикулярности двух прямых.

are_similar – проверка двух треугольников на подобие.

are_tangent – проверка, касательна ли прямая сфере.

coplanar – определение принадлежности четырех точек одной плоскости.

on_plane(pt3d,plane) – проверка принадлежности точки **pt3d** плоскости **plane**.

on_sphere(pt3d,sphere) – проверка принадлежности точки **pt3d** сфере **sphere**.

В следующих командах, если это не оговорено особо, результат действия присваивается переменной с именем **name**. Задание большинства геометрических объектов (вершины треугольника или тетраэдра, точки отрезка и др.) не должно содержать никаких символических переменных.

angle – вычисление наименьшего угла между двумя прямыми.

area – вычисление площади треугольника.

center – вычисление центра сферы.

coordinates – выдача координат точки.

distance – вычисление расстояния между двумя точками.

inter – вычисление точки пересечения двух прямых.

midpoint – вычисление прямой, проходящей через заданные две точки.

parallel(pt3d,plane,name) – вычисление плоскости, проходящей через точку **pt3d** и параллельной плоскости **plane**.

perpendicular(pt3d,plane,name) – вычисление плоскости, проходящей через точку **pt3d** и перпендикулярной плоскости **plane**.

projection – вычисление проекции точки на плоскость.

radius – вычисление радиуса сферы.

reflect(pt3d,plane,name) – вычисление точки, зеркально симметричной точке **pt3d** относительно плоскости **plane**.

sphere(name,[pt3d,expr]) – определение сферы с центром в точке **pt3d** и радиусом **expr**.

symmetric(pt3d,plane,name) – вычисление точки, симметричной точке **pt3d** относительно плоскости **plane**.

tangent(pt3d,sphere,name) – вычисление касательной плоскости к сфере **sphere**, проходящей через точку **pt3d**.

tetrahedron – вычисление тетраэдра по четырем точкам.

triangle3d – вычисление треугольника по трем точкам.

volume – вычисление объема сферы.

Приведем пример действия некоторых команд. Зададим сферу S, вычислим ее радиус и объем, а затем проверим, принадлежит ли ей точка A. Как видно, вычисленные радиус и объем зависят от символьной константы a, а результатом последней команды является условие принадлежности точки сфере.

```
> with(geom3d):
> sphere(S,[x^2+y^2+z^2-a=0]);
```

$$S$$

```
> radius(S); volume(S);
```

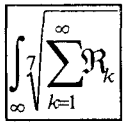
$$\sqrt{a}$$

$$\frac{4}{3}\pi a^{3/2}$$

```
> on_sphere(point3d(A,[-1,0,1]),S);
```

THE POINT IS ON THE SPHERE IF:

$$2-a=0$$



4. Математический анализ

В этой главе изложены команды Maple, реализующие основные математические операции, известные читателям в основном из одноименного курса в вузе. Мы предполагаем, что человек, взявший в руки эту книгу, имеет за плечами по крайней мере один курс университета или технического вуза и, следовательно, знаком с задачниками Демидовича, Кудрявцева и других авторов. Примерами из книг Демидовича для университетов [3] и втузов [4] мы и будем иллюстрировать работу Maple, обозначая примеры из первой книги буквой *Д*, а из второй – *д*, ставя после буквы номер упражнения.

В Maple для некоторых математических операций существует по две команды: одна прямого, а другая – отложенного исполнения, причем имена этих команд состоят из одинаковых букв. Команды прямого исполнения начинаются с маленькой буквы и выполняются немедленно. Отложенные команды начинаются с большой буквы. После обращения к отложенной команде заданная математическая операция (интеграл, производная, предел и т.д.) выводится в стандартном математическом виде и сразу не вычисляется. Для выполнения отложенной операции нужно использовать команду **value**. Перед использованием некоторых команд необходимо их предварительно вызвать из стандартной библиотеки при помощи операции **readlib** или подключить соответствующую библиотеку при помощи **with**.

Часто для задания области определения функции или в других случаях необходимо наложить условия на переменные, и в Maple это легко сделать, используя команду **assume(rel)**, где **rel** – логическое выражение. Например, после команды

```
> assume(a>0);
```

везде, где будет встречаться переменная **a**, она будет обозначаться **a~** и считаться положительной.

4.1. Пределы, суммы, ряды

Обычно курс математического анализа начинается с понятия предела последовательности и функции. Для вычисления пределов в

Maple существуют команды **limit(expr,x=val,dir)** и **Limit(expr,x=val,dir)**, здесь **expr** – выражение, для которого вычисляется предел (функция или n-й член последовательности), **x=val** – значение точки, для которой вычисляется предел, а **dir** – необязательный параметр, который может принимать следующие значения: **left** (предел слева), **right** (предел справа), **real** (действительный) или **complex** (комплексный). Проиллюстрируем сказанное примерами:

д180. Найти предел: $\lim\limits_{n\to\infty}\dfrac{n\sin(n!)}{n^2+1}$.

```
> Limit(n*sin(n!)/(n^2+1),n=infinity); value(");
```

$$\lim_{n\to\infty}\frac{n\sin(n!)}{n^2+1}$$

$$0$$

Д540.1. Найти предел: $\lim\limits_{x\to 0}\dfrac{\ln\left(nx+\sqrt{1-n^2x^2}\right)}{\ln\left(x+\sqrt{1-x^2}\right)}$.

```
> limit(ln(n*x+sqrt(1-n^2*x^2))
>       /ln(x+sqrt(1-x^2)),x=0);
```

$$n$$

Д596. Найти: а) $\lim\limits_{x\to-0}\dfrac{1}{1+e^{\frac{1}{x}}}$, *b)* $\lim\limits_{x\to+0}\dfrac{1}{1+e^{\frac{1}{x}}}$.

```
> limit(1/(1+exp(1/x)),x=0,left);
```

$$1$$

```
> limit(1/(1+exp(1/x)),x=0,right);
```

$$0$$

Для операции суммирования используются команды **sum(expr,var=var1..var2)** и **Sum(expr,var=var1..var2)**, где **expr** – выражение, зависящее от переменной суммирования **var**, а **var1..var2** – пределы суммирования. Различие между этими двумя командами заключается в том, что в варианте, начинающемся с маленькой буквы, суммирование проводится сразу, а для **Sum** нужно дополнительно дать команду **value**. Пределы суммирования могут быть как конечными, так и бесконечными и эта

команда может быть использована также и для суммирования рядов. Например:

```
> sum(2^n*(n+1)/n!,n=0..30);
```

$$\frac{876174090007275541491252 01}{395257562119053391570312 5}$$

Д3012. Найти сумму ряда: $\sum_{n=1}^{\infty} n(n+2)x^n$.

```
> Sum(n*(n+2)*x^n,n=1..infinity);
```

$$\sum_{n=1}^{\infty} n\,(n+2)\,x^n$$

```
> simplify(value(%));
```

$$\frac{(x-3)\,x}{(x-1)^3}$$

Аналогичным образом для вычисления бесконечных и конечных произведений используются две команды: **product(expr,k)** и **Product(expr,k=k1..k2)**, здесь **expr** – k-й член произведения, **k** – индекс, а **k1** и **k2** задают интервал изменения индекса. Например:

Д3051. Доказать равенство: $\prod_{n=2}^{\infty}\left(1-\frac{1}{n^2}\right)=\frac{1}{2}$.

```
> limit(product(1-1/n^2,n=2..k),k=infinity);
```

$$\frac{1}{2}$$

4.2. Исследование функций

Как известно, исследование функции необходимо начинать с выяснения ее области определения, но, к сожалению, это трудно автоматизируемая операция, и решение данной задачи не может быть пока полностью переложено на плечи Maple. Обычно при рассмотрении этого вопроса приходится решать неравенства, а с этим, как показано в гл. 3 "Элементарная математика", этот пакет довольно успешно справляется.

Для исследования экстремумов функции как одной, так и многих переменных используется команда **extrema(expr,constr, vars,nv)**, где **expr** – выражение, экстремумы которого нужно найти, **constr** – ограничения, **vars** – переменные, по которым разыскивается экстремум, а **nv** – имя переменной, которой будут присвоены координаты точек экстремумов. Перед обращением к функции **extrema** ее необходимо вызвать из стандартной библиотеки командой **readlib**. Соответствующий пример:

Д1442. Найти экстремум функции $y = \arctan(x) - \frac{1}{2}\ln(1+x^2)$.

```
> readlib(extrema):
> extrema(arctan(x)-ln(1+x^2)/2, {} ,x,'s'); s;
```

$$\left\{\frac{1}{4}\pi - \frac{1}{2}\ln(2)\right\}$$

$$\{\{x = 1\}\}$$

Для поиска минимума и максимума функции используются соответственно команды: **minimize(expr,vars,ranges)** и **maximize(expr,vars,ranges)**, где **expr** – выражение, **vars** – переменные, по которым ищется минимум или максимум, а **ranges** – интервал изменения переменных, причем здесь может стоять строка **'infinite'**, т.е. минимум или максимум будет разыскиваться на всей числовой оси.

Проверить непрерывность алгебраического выражения, зависящего от переменной **x**, на отрезке **[x1,x2]** можно при помощи команды **iscont(expr,x=x1..x2)**; результатом будет булевская константа 'истина' (**true**) или 'ложь' (**false**). Для определения точек, в которых нарушается непрерывность выражения **expr** по переменной **x**, используется команда **discont(expr,x)**.

д323. Исследовать на непрерывность функцию $y = \ln(\cos x)$.

```
> readlib(iscont): readlib(discont):
> iscont(ln(cos(x)),x = -infinity..+infinity);
> discont(ln(cos(x)),x);
```

false

$$\left\{\frac{1}{2}\pi + \pi\_Z1\sim\right\}$$

Здесь *_Z1~* означает любое целое положительное число.

Кроме того, чтобы найти точки разрыва функции, можно обратиться к команде **singular(expr,vars)**, здесь **expr** – выражение, зависящее от переменных **vars**, причем **expr** может зависеть и от других переменных, например:

```
> readlib(singular): singular(tan(x/(x-y)),x);
```

$$\left\{x = -\frac{-2\_N\pi y - \pi y}{-2 + 2\_N\pi + \pi}\right\}, \{x = y\}$$

Здесь *_N* означает любое целое число.

Вычет алгебраического выражения **expr** при **x=a** вычисляется при помощи команды **residue(f,x=a)**.

4.3. Дифференцирование и интегрирование

Вычисление производной выражения **expr** по переменной **x** осуществляется при помощи команд **diff(expr,x)** или **Diff(expr,x)**. Отметим, что эти команды могут использоваться и для вычисления частных производных функций многих переменных, в этом случае используется следующий их формат: **diff(expr,x1$n1,x2$n2,...)**, здесь **expr** – выражение, зависящее от переменных **x1**,**x2**,..., а **n1**,**n2**,... – порядки дифференцирования по соответствующим переменным. Для команды **Diff** параметры аналогичны. Операция **diff** используется и во многих других случаях (например, см. гл. 6 "Дифференциальные уравнения").

Д523. Найти производную функции: $y = \frac{1}{2}\ln\tan\frac{x}{2} - \frac{1}{2}\frac{\cos x}{\sin^2 x}$.

```
> Diff(ln(tan(x/2))/2-cos(x)/sin(x)^2/2,x);
```

$$\frac{\partial}{\partial x}\left(\frac{1}{2}\ln\left(\tan\left(\frac{1}{2}x\right)\right) - \frac{1}{2}\frac{\cos(x)}{\sin(x)^2}\right)$$

```
> simplify(value(")");
```

$$-\frac{1}{\left(\cos(x)^2 - 1\right)\sin(x)}$$

Д1813. Найти частные производные функции $u = z^{xy}$. *(Дополним этот пример вычислением второй производной по* z.*)*

```
> u:=z^(x*y): diff(u,x); diff(u,y); diff(u,z);
```

$$z^{(x\,y)}\,y\ln(z)$$

$$z^{(x\,y)}\,x\ln(z)$$

$$\frac{z^{(x\,y)}\,x\,y}{z}$$

```
> factor(diff(u,z$2));
```

$$\frac{z^{(x\,y)}\,x\,y\,(x\,y-1)}{z^2}$$

Для задания дифференциального оператора используется символ D. Чтобы преобразовать выражение с оператором D к виду, использующему команду **diff**, применяется команда **convert**. Она же используется и для обратного преобразования. Поясним сказанное следующими примерами:

```
> D(sin); D(ln-arcsin);
```

$$\left(a \to \frac{1}{a}\right) - \left(a \to \frac{1}{\sqrt{1-a^2}}\right)$$

```
> f:=diff(y(x),x$7); convert(f,D);
```

$$f := \frac{\partial^7}{\partial x^7}\,y(x)$$

$$D^{(7)}(y)(x)$$

Теперь перейдем к интегрированию функций. Для этих целей в пакете предусмотрено несколько команд, находящихся в различных библиотеках. В стандартной библиотеке находятся процедуры **int(expr,par)** и **Int(expr,par)**, которые в зависимости от параметров **par** могут использоваться для поиска неопределенных интегралов, аналитического или численного вычисления определенных, собственных и несобственных интегралов. Для поиска неопределенных интегралов используется следующая команда:

int(expr,var),

здесь **expr** – интегрируемое выражение, а **var** – переменная интегрирования. Для отложенной команды **Int** все аналогично.

Д2165. Найти интеграл: $\int \frac{1+\sin x}{1+\cos x} e^x dx$.

```
> int((1+sin(x))*exp(x)/(1+cos(x)),x);
```

$$e^x \tan\left(\frac{1}{2}x\right)$$

Для вычисления определенных интегралов используется команда **int(expr,var=a..b)**, где **expr** – интегрируемое выражение, **var** – переменная интегрирования и **(a,b)** – отрезок интегрирования. Для вычисления двойных, тройных и т.д. интегралов нужно применить эту команду несколько раз. Поясним это примерами:

Д2280. Вычислить интеграл: $\int_0^{\ln 2} \sinh^4 x dx$.

```
> int(sinh(x)^4,x=0..ln(2));
```

$$-\frac{225}{1024}+\frac{3}{8}\ln(2)$$

д2117. Вычислить повторный интеграл: $\int_{-3}^{3} dy \int_{y^2-4}^{5} (x+2y)dx$.

```
> Int(Int(x+2*y,x=y^2-4..5),y=-3..3);
```

$$\int_{-3}^{3}\int_{y^2-4}^{5} x+2y\,dx\,dy$$

```
> evalf(value(%));
```

$$50.40000000$$

Чтобы посчитать интеграл численно, необходимо выполнить команду **evalf(int(expr),x=a..b,digits,flag)**. Здесь обязательными параметрами являются подинтегральная функция **expr**, зависящая только от переменной **x**, **a** и **b** – пределы интегрирования, а необязательными – **digits** – точность вычисления (по умолчанию равна константе **Digits**) и **flag** – код численного метода.

Несколько команд интегрирования имеется в библиотеке **student**, которую предварительно нужно подключить при помощи

with(student). Отметим, что все эти команды отложенного действия. Кратко их перечислим:

Int(expr,x) – интегрирование; здесь **expr** – интегрируемое выражение, а **x** – переменная интегрирования;

Doubleint(expr,x,y,Domain) – двойное интегрирование выражения **expr** по переменным **x** и **y** в области **Domain**;

Lineint(f(x,y),x,y) – вычисление линейного интеграла. Переменная **x** считается зависящей от переменной **y**, а если переменных больше, то все они считаются зависящими от последней;

Tripleint(g,x,y,z) – вычисление тройного интеграла;

changevar(s,f,u) – замена переменных, где **s** – выражение, задающее замену координат, **f** – одна из перечисленных команд интегрирования из пакета **student**, а **u** – список новых переменных интегрирования;

intparts(f,u) – интегрирование по частям, здесь **f** – выражение **Int**[**student**], а **u** – часть подинтегрального выражения, которая будет дифференцироваться.

Поясним некоторые из этих команд примерами.

∂1585. Применяя подстановку $\tan\left(\frac{x}{2}\right)=t$, *вычислить интеграл* $\int_0^{\pi}\frac{dx}{3+2\cos x}$

```
> with(student):
> changevar(tan(x/2)=t,Int(1/(3+2*cos(x)),
>           x=0..Pi),t);
```

$$\int_0^{\infty} 2\,\frac{1}{(3+2\cos(2\arctan(t)))\left(1+t^2\right)}\,dt$$

```
> value(");
```

$$\frac{1}{5}\sqrt{5}\,\pi$$

∂2245. Вычислить интеграл $\int_0^2 dx\int_0^{2\sqrt{x}} dy\int_0^{\sqrt{\frac{4x-y^2}{2}}} xdz$.

```
> Tripleint(x,z=0..sqrt((4*x-y^2)/2),
```

```
y=0..2*sqrt(x),x=0..2);
```

$$\int_0^2 \int_0^{2\sqrt{x}} \int_0^{\frac{1}{2}\sqrt{8x-2y^2}} x \, dz \, dy \, dx$$

```
> simplify(value(")) ;
```

$$\frac{4}{3}\sqrt{2}\,\pi$$

Д2240. Найти интеграл $\int_0^{\pi} x \sin x \, dx$.

```
> intparts(Int(x*sin(x), x=0..Pi), x); value(");
```

$$\pi - \int_0^{\pi} -\cos(x) \, dx$$

$$\pi$$

4.4. Разложение и приближение функций

При операциях разложения функций в ряды порядок разложения по умолчанию определяется значением зарезервированной константы **Order**, которая может переопределяться пользователем в процессе работы.

Команда **series(expr, eqn, n)** выдает разложение выражения **expr** в степенной ряд, здесь **eqn** – выражение вида $x=a$, где x – переменная разложения, a – точка, в окрестности которой выписывается разложение, **n** – порядок. Приведем соответствующий пример:

Д2848. Написать три члена разложения функции $f(x)=(1+x)^{\frac{1}{x}}$.

```
> series((1+x)^(1/x),x=0,3);
```

$$e - \frac{1}{2}e\,x + \frac{11}{24}e\,x^2 + O(x^3)$$

Теперь, следуя задачнику, перейдем к ряду Тейлора. Для разложения выражения expr, зависящего от переменной **x**, в ряд Тейлора в точке **x=x0** до членов порядка **n**, можно воспользоваться

командой **taylor(expr,x=x0,n)**, причем если последний параметр не указан, то порядок разложения определяется значением константы **Order**.

Д1383. Написать разложение функции $\sqrt[3]{\sin x^3}$ *до члена с* x^{13}.

```
> Order:=14; taylor(sin(x^3)^(1/3),x,14);
```

$$x-\frac{1}{18}x^7-\frac{1}{3240}x^{13}+O\left(x^{14}\right)$$

Выражения, зависящие от нескольких переменных, разлагаются в ряд Тейлора при помощи команды **mtaylor(expr,vars,k,w)**; здесь **expr** – разлагаемое в ряд выражение, **vars** – список имен переменных разложения, **k** – порядок разложения, а **w** – вес. Команду необходимо предварительно подгрузить, вызвав **readlib(mtaylor)**. Проиллюстрируем действие команды на следующем примере:

д2669. Написать четыре первых члена разложения по степеням x и y функции $e^x\cos y$.

```
> readlib(mtaylor):mtaylor(exp(x)*cos(y),[x=0,y],5);
```

$$1+x-\frac{1}{2}y^2+\frac{1}{2}x^2+\frac{1}{6}x^3-\frac{1}{2}xy^2+\frac{1}{24}y^4-\frac{1}{4}x^2y^2+\frac{1}{24}x^4$$

Практически аналогична предыдущей по своему действию команда **poisson(f,v,n,w)** и отличается представлением тригонометрических функций в разложении.

Если нужно получить только коэффициент при члене порядка **k**, то лучше воспользоваться командой **coeftayl(expr,vars,k)**, где **vars** – список имен переменных разложения, **k** – порядок члена разложения, коэффициент при котором выписывается. Для функции нескольких переменных параметр **k** является списком.

Для разложения выражения **expr** в ряд Лорана можно использовать команду **laurent(expr,x=a,n)** из библиотеки **numapprox** с параметрами, которые аналогичны параметрам команды **taylor**.

При помощи Maple можно решать задачи приближения функций. Для разложения выражения в ряд по полиномам Чебышева в пакете существует команда **chebyshev**. Команда **asympt(f,x,n)** позволяет выписать асимптотическое разложение выражения **f** по

степеням переменной **x** порядка **n**, когда **x** стремится к бесконечности.

Естественно, описанные команды не исчерпывают всех возможностей Maple в области математического анализа. Большое число команд, и даже библиотек, написано многочисленными пользователями и распространяется по компьютерным сетям. Освоение этого материала требует затрат времени, и иногда оказывается проще написать свои программы, реализующие те или иные операции. В качестве примера в гл. 10 "Мини-исследования" приведена команда разложения функции в ряд Фурье.

5. Линейная алгебра

После описания возможностей пакета Maple для решения задач математического анализа естественно перейти к рассмотрению другого классического математического курса – линейной алгебры. Мы не ставим своей целью подробно рассказать здесь о всех имеющихся в Maple командах линейной алгебры, а представим только краткий обзор основных и наиболее употребимых с нашей точки зрения. Небольшая часть команд для решения задач линейной алгебры содержится в стандартной библиотеке и пакете **student**, большая же часть находится в библиотеке **linalg**.

Основными объектами команд линейной алгебры являются матрицы и векторы. Матрицей в Maple считается двумерный массив, у которого индексы изменяются от единицы, так что, например, переменная, описанная как **array(0..3,1..4)**, матрицей не является. Можно использовать два способа задания матриц: при помощи команд-описателей **matrix** и **array**. Коротко формат описателя **array** уже обсуждался в гл.1 "Среда Maple", дополнительно отметим, что при помощи этой команды можно задавать матрицы специального вида. Параметры **symmetric** и **antisymmetric** указываются для определения симметричных и кососимметричных матриц, а для определения единичных и диагональных матриц используются соответственно параметры **identity** и **diagonal**.

Команда-описатель **matrix** из библиотеки линейной алгебры имеет вид:

```
matrix(n,m,[val1,val2,...])
```

здесь **n** – число строк, **m** – число столбцов матрицы, а **val1**,**val2**,... – значения элементов матрицы. Вместе с тем существуют и другие формы этого описателя, например: **matrix(n,m,f)**, здесь **f** – функция от двух целых переменных (индексов матрицы), с помощью которой присваиваются значения элементам матрицы.

Вектором в Maple считается одномерный массив, который можно определить при помощи описателя **array(1..n,[val1, val2,...])**, где **n** – его размерность, а **val1**, **val2**,... – значения элементов. Другой способ описания вектора использует команду **vector** из пакета **linalg**:

```
vector(n,[val1,val2,...])
```

здесь **n** – размерность вектора, а **val1**, **val2**...– значения элементов, причем если опустить параметр размерности, то размерность вектора определяется числом его элементов. Существует также другой вариант этого описателя: **vector(n,f)**; здесь **f** – функция от индекса, используемая для генерации вектора.

Значения элементов матриц и векторов можно задавать как при описании, так и в ходе работы при помощи оператора присваивания, при этом часть элементов массива (матрицы, вектора) можно не определять. В последнем случае Maple заполнит вакансии переменными по умолчанию. Напомним, что для просмотра содержимого составных переменных нужно использовать команду **eval** или **op**.

При иллюстрации действия команд в этой главе мы будем считать, что результаты ранее приводимых примеров сохраняются для последующих.

Приведем несколько примеров задания матриц и векторов.

```
> A:=matrix(4,3,[[a,b,c],[],[1,2,3],[x,y,z]]);
```

$$A := \begin{bmatrix} a & b & c \\ A_{2,1} & A_{2,2} & A_{2,3} \\ 1 & 2 & 3 \\ x & y & z \end{bmatrix}$$

```
> fun:=(i,j)->x^i/y^j; B:=matrix(2,3,fun);
```

$$fun := (i,j) \to \frac{x^i}{y^j}$$

$$B := \begin{bmatrix} \frac{x}{y} & \frac{x}{y^2} & \frac{x}{y^3} \\ \frac{x^2}{y} & \frac{x^2}{y^2} & \frac{x^2}{y^3} \end{bmatrix}$$

```
> u:=array(1..3); v:=vector([a,b,c]);
```

$$u := \text{array}(1..3, [\,])$$

$$v := [a \quad b \quad c]$$

```
> F:=array(1..3,1..3,antisymmetric): eval(F);
```

$$\begin{bmatrix} 0 & ?_{1,2} & ?_{1,3} \\ -?_{1,2} & 0 & ?_{2,3} \\ -?_{1,3} & -?_{2,3} & 0 \end{bmatrix}$$

В последнем примере знак вопроса означает неопределенность элементов матрицы.

Кроме того, объекты линейной алгебры (матрицы, векторы) могут быть получены как результат преобразования массивов, списков, таблиц и т.д. Например, команда **convert(list,vector)** преобразует переменную типа **list** в вектор, а для преобразования набора переменных типа **list** в матрицу используется команда **convert([list1,...,listn],matrix)**.

Существуют также команды, позволяющие формировать матрицы специального вида. Так, команда **hilbert(n,expr)** генерирует гильбертову матрицу размерности n×n для выражения expr. Для задания блочной матрицы используется команда **blockmatrix**, случайная матрица задается командой **randmatrix**.

В Maple реализованы все операции линейной алгебры. Ниже приводится описание Maple-команд для большей части возможных операций. Более подробную информацию о командах библиотеки **linalg** можно получить, обратившись к справке: **?linalg**.

5.1. Работа со структурой матрицы и вектора

Перечислим некоторые команды, позволяющие получать информацию о матричных и векторных переменных, корректировать эти объекты и работать с отдельными строками и столбцами.

Для выяснения размерностей матрицы M имеются две команды: **rowdim(M)** — выдает число строк матрицы M и **coldim(M)** – число столбцов. Например:

```
> rowdim(A); coldim(A);
                              4
                              3
```

Число элементов вектора v можно узнать при помощи команды **vectdim(v)**.

Далее перечислим команды, позволяющие изменять размерности матриц, добавлять или удалять строки и столбцы. Для того чтобы удалить из матрицы M строки с номерами от i до j, нужно воспользоваться командой **delcols(M,i..j)**, а для удаления столбцов применяется команда **delrows(M,i..j)**. Приведем пример удаления второй строки из матрицы A:

```
> C:=delrows(A,2..2);
```

$$C := \begin{bmatrix} a & b & c \\ 1 & 2 & 3 \\ x & y & z \end{bmatrix}$$

Увеличить размерность матрицы **M** можно при помощи команды **extend(M,m,n,x)**; здесь **m** – число добавляемых строк, **n** – число добавляемых столбцов, **x** – значение заполнителя. Для копирования матрицы **A** в матрицу **B**, начиная с элемента с номером **i,j**, используется команда **copyinto(A,B,i,j)**. При помощи команды **concat(M1,M2)** можно получить новую матрицу, являющуюся горизонтальным слиянием двух матриц **M1** и **M2**, имеющих одинаковое число строк. Для вертикального слияния матриц **A** и **B** используется команда **stack(A,B)**. Например:

```
> G:=concat(C,F);
```

$$G := \begin{bmatrix} a & b & c & 0 & F_{1,2} & F_{1,3} \\ 1 & 2 & 3 & -F_{1,2} & 0 & F_{2,3} \\ x & y & z & -F_{1,3} & -F_{2,3} & 0 \end{bmatrix}$$

Чтобы выделить (без удаления) из матрицы M строку с номером **i**, используется команда **row(M,i)**, а для выделения столбца с номером **i** используется команда **col(M,i)**. Для формирования подматрицы, включающей элементы столбцов с номерами от **i1** до **i2** и строк с номерами от **j1** до **j2**, предусмотрена команда **submatrix(M,i1..i2,j1..j2)**. Команда **subvector(v,i..j)** выделяет подвектор с элементами **v[i]**,...,**v[j]** из вектора **v**. Выделение минора матрицы **M** для элемента с индексами **i,j** осуществляется командой **minor(M,i,j)**. Приведем пример выделения подматрицы:

```
> submatrix(G,2..3,3..4);
```

$$\begin{bmatrix} 3 & -F_{1,2} \\ z & -F_{1,3} \end{bmatrix}$$

Теперь опишем некоторые команды, позволяющие оперировать со строками и столбцами матрицы. Так, команда **addcol(M,i1,i2,expr)** формирует новую матрицу, получаемую из матрицы **M** прибавлением к столбцу с номером **i2** столбца с номером **i1**, умноженного на выражение **expr**. Аналогичная команда для строк имеет вид: **addrow(M,i1,i2,expr)**. Для умножения столбца матрицы **M** с номером **i** на выражение **expr** применяется команда **mulcol(M,i,expr)**. Аналогичная команда для умножения строки имеет вид: **mulrow(M,i,expr)**. Чтобы переставить местами строки матрицы **M** с номерами **i1** и **i2**, нужно выполнить команду **swaprow(M,r1,r2)**, а для перестановки столбцов используется команда **swapcol(A, c1, c2)**.

5.2. Основные матричные и векторные операции

Перейдем к командам, которые реализуют основные векторные и матричные операции. Начнем с самых простых и часто необходимых операций линейной алгебры – действий с самими матрицами и векторами.

Для сложения двух матриц (векторов) **A** и **B** одинаковой размерности можно использовать две команды: **add(A,B)** или **evalm(A+B)**, причем для команды **add** существует расширенный вариант: **add(A,B,c,d)** — сложение матриц (векторов) **A** и **B** со скалярными множителями **c** и **d**, т.е. вычисляется выражение **(c*A+d*B)**.

Умножить матрицу **A** на матрицу (вектор) **B** тоже можно двумя способами: **multiply(A,B)** или **evalm(A&*B)**. Проиллюстрируем сказанное примерами:

```
> add(C,F,x,y); evalm(C&*u);
```

$$\begin{bmatrix} x\,a & x\,b+y\,F_{1,2} & x\,c+y\,F_{1,3} \\ x-y\,F_{1,2} & 2\,x & 3\,x+y\,F_{2,3} \\ x^2-y\,F_{1,3} & x\,y-y\,F_{2,3} & x\,z \end{bmatrix}$$

$$\begin{bmatrix} a u_1 + b u_2 + c u_3 & u_1 + 2 u_2 + 3 u_3 & x u_1 + y u_2 + z u_3 \end{bmatrix}$$

Возведение матрицы **M** в степень **n** осуществляется командой: **evalm(M^n)**. Обратную матрицу к матрице **M** можно вычислить также двумя способами: **inverse(M)** или **evalm(1/M)**. Транспонировать матрицу M можно при помощи команды **transpose(M)**. Вычислить эрмитову транспонированную матрицу можно командой **htranspose(A)**. Для вычисления сопряженной матрицы используются команды **adjoint(M)** и **adj(M)**. Приведем несколько примеров:

```
> adjoint(C);
```

$$\begin{bmatrix} 2z - 3y & -bz + cy & 3b - 2c \\ -z + 3x & az - xc & -3a + c \\ y - 2x & -ay + xb & 2a - b \end{bmatrix}$$

```
> transpose(F);
```

$$\begin{bmatrix} 0 & -F_{1,2} & -F_{1,3} \\ F_{1,2} & 0 & -F_{2,3} \\ F_{1,3} & F_{2,3} & 0 \end{bmatrix}$$

Чтобы вычислить определитель матрицы **M**, достаточно выполнить команду **det(M)** из пакета **linalg** или отложенную команду **Det(M)** из стандартной библиотеки. Для вычисления числа обусловленности используется команда **cond(M)**, а для вычисления следа – команда **trace(M)**. Ранг матрицы **M** вычисляется командой **rank(M)**. Например:

```
> det(F); rank(F); trace(C);
```

$$0$$

$$2$$

$$a + 2 + z$$

5.3. Решение задач линейной алгебры

Этот раздел начнем с изложения команд, позволяющих изучить спектр квадратной матрицы **M**. В стандартной библиотеке

для поиска собственных чисел и собственных векторов числовой матрицы существует команда с отложенным исполнением

Eigenvals(M,vects);

здесь **M** – квадратная числовая матрица, а **vects** – необязательный параметр, наличие которого говорит о том, что кроме собственных чисел вычисляются и собственные векторы. Результатом действия команды являются собственные числа, а соответствующие собственные векторы будут находиться в колонках матрицы **vects**. Приведем пример обращения к данной команде:

```
> AA:=array(1..2,1..2,[[23/25,36/25],[36/25,2/25]]):
> vv:=evalf(Eigenvals(AA,W)); eval(W);
```

$$vv := [-1.0000000 \quad 2.0000000]$$
$$\begin{bmatrix} .60000000 & -.80000000 \\ -.80000000 & -.60000000 \end{bmatrix}$$

Для исследования спектра символьной матрицы нужно использовать команды из библиотеки **linalg**. Для вычисления собственных чисел матрицы **M** используется команда **eigenvals(M)**, и результатом ее действия является массив, содержащий собственные числа. Для поиска собственных чисел и собственных векторов применяется команда **eigenvects(M)**, причем результат выдается в виде массива, каждая строка которого состоит из собственного числа, его кратности и соответствующего собственного вектора. Поясним сказанное примером:

```
> CC:=array(1..3,1..3,[[x,0,y],[x,y,0],[y,0,x]]);
```

$$CC := \begin{bmatrix} x & 0 & y \\ x & y & 0 \\ y & 0 & x \end{bmatrix}$$

```
> linalg[eigenvects](CC);
```

$$[x+y, 1, \{[1 \quad 1 \quad 1]\}],$$
$$\left[x-y, 1, \left\{\left[\frac{x-2y}{x} \quad 1 \quad -\frac{x-2y}{x}\right]\right\}\right],$$
$$[y, 1, \{[0 \quad 1 \quad 0]\}]$$

Для вычисления характеристического многочлена матрицы **M** относительно переменной λ используется команда

charpoly(M,lambda). Выяснить положительную или отрицательную определенность матрицы можно при помощи команды **definite(M,kind)**. Здесь **M** – квадратная матрица, а **kind** – параметр, который может принимать значения **'positive_def'**, **'positive_semidef'**, **'negative_def'** и **'negative_semidef'**. Результатом действия команды будет булевская константа (**true** или **false**). Ядро матрицы **M** вычисляется при помощи команды **kernel(M)**. Например:

```
> F:=array(1..3,1..3,antisymmetric): kernel(F);
```

$$\left\{\begin{bmatrix} 1 & -\dfrac{F_{1,3}}{F_{2,3}} & \dfrac{F_{1,2}}{F_{2,3}} \end{bmatrix}\right\}$$

В Maple реализованы практически все алгоритмы приведения матриц к различным специальным формам. Для приведения матрицы **M** к жордановой форме используется команда **jordan(M)**. Результатом вызова команды **gausselim(M)** будет матрица, приведенная к треугольному виду. Применить к матрице **M** алгоритм гауссова исключения без деления можно командой **ffgausselim(M)**. Команда приведения матрицы **M** к треугольному виду при помощи алгоритма Гаусса–Жордана называется **gaussjord**. Чтобы получить эрмитову форму матрицы **M**, элементы которой зависят от переменной **x**, нужно обратиться к команде **hermite(M,x)**. Приведем два примера:

```
> C:=array(1..3,1..3,[[a,b,c],[1,2,3],[x,y,z]]):
> gaussjord(C);
```

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
> ffgausselim(C);
```

$$\begin{bmatrix} a & b & c \\ 0 & 2a-b & 3a-c \\ 0 & 0 & -3ya+2az+3bx-bz-2xc+yc \end{bmatrix}$$

Отметим, что для работы с символьными матрицами команда **ffgausselim** предпочтительнее, поскольку не производит нормировку элементов и исключает возможные ошибки, связанные с делением на нуль.

Следующая группа команд позволяет изучать векторное пространство, порождаемое матрицей **M**. Так, для определения размерно-

сти векторного пространства, порожденного столбцами матрицы, используется команда **colspace(M)**, а аналогичная команда для строк называется **rowspace(M)**. Чтобы проверить ортогональность матрицы **M**, достаточно вызвать команду **orthog(M)**. Например:

```
> fun:=(i,j)->x^i/y^j: B:=matrix(2,3,fun);
```

$$B := \begin{bmatrix} \frac{x}{y} & \frac{x}{y^2} & \frac{x}{y^3} \\ \frac{x^2}{y} & \frac{x^2}{y^2} & \frac{x^2}{y^3} \end{bmatrix}$$

```
> colspace(B);
```

$$\{[1 \quad x]\}$$

Теперь перейдем к векторным операциям. Нормировать вектор **v** можно при помощи команды **normalize(v)**. Чтобы вычислить угол между двумя векторами **v** и **u**, нужно выполнить команду **angle(v,u)**. При помощи команды **GramSchmidt([v1,v2,...])** можно сформировать ортогональный базис векторного пространства, генерируемого линейно-независимыми числовыми векторами **v1**,**v2**,... Команда **basis([v1,v2,...vk])** определяет базис для набора векторов **v1**,**v2**,...,**vk**. Приведем иллюстрирующий последние две команды пример:

```
> v1:=vector([1,2,3]): v2:=vector([-1,2,3]):
> v3:=vector([11,12,13]): v4:=vector([1,1,1]):
> bas:=basis([v1,v2,v3,v4]);
```

$$bas := [v1, v2, v3]$$

```
> GramSchmidt(bas);
```

$$\left[[1 \quad 2 \quad 3], \left[\frac{-13}{7} \quad \frac{2}{7} \quad \frac{3}{7}\right], \left[0 \quad \frac{30}{13} \quad \frac{-20}{13}\right]\right]$$

Кроме того, в пакете **linalg** имеются команды решения систем линейных уравнений, отличные от команды **solve** из стандартной библиотеки. Для решения системы линейных алгебраических уравнений $Mx = B$, где M – матрица, а B – вектор или матрица правых частей, существуют команды **linsolve(M,B)** и **leastsqrs(M,B)** (метод наименьших квадратов). Приведем пример решения системы линейных уравнений:

```
> CC:=array(1..3,1..3,[[x,0,y],[x,y,0],[y,0,x]]);
> linalg[linsolve](CC, v);
```

$$\left[\begin{array}{ccc} \dfrac{-y\,c+a\,x}{-y^2+x^2} & \dfrac{-b\,y^2+b\,x^2+y\,x\,c-a\,x^2}{(-y^2+x^2)\,y} & \dfrac{x\,c-y\,a}{-y^2+x^2} \end{array}\right]$$

5.4. Векторный анализ

Скалярное произведение двух векторов **u** и **v** вычисляется командой **dotprod(u,v)**, а векторное произведение – командой **crossprod(u,v)**. Для вычисления нормы векторов и матриц используется команда **norm(M)**. Приведем примеры:

```
> w:=crossprod(u,[a,b,c]);
```

$$w := \left[u_2\,c - u_3\,b \quad u_3\,a - u_1\,c \quad u_1\,b - u_2\,a\right]$$

```
> simplify(dotprod(u,w));
```

$$0$$

Перейдем теперь к дифференциальным операторам векторного анализа. Чтобы вычислить градиент скалярной функции **f**, зависящей от набора переменных **X**, нужно выполнить команду **grad(f,X)**. Дивергенцию векторной функции **F**, зависящей от набора переменных **X**, можно получить при помощи команды **diverge(F,X)**. Например:

```
> f:=x^2+y^2+sin(z):
> grd:=grad(f,[x,y,z]);
```

$$grd := [\,2\,x,\ 2\,y,\ \cos(z)\,]$$

```
> diverge(grd,[x,y,z]);
```

$$4 - \sin(z)$$

И еще три важных дифференциальных оператора и соответствующие Maple-команды:

curl(v,x) – вычисление ротора трехмерного вектора **v** по трем переменным **x**;

laplacian(f,x) – вычисление лапласиана функции **f** по переменным **x**;

jacobian(v,x) – вычисление матрицы Якоби для вектора **v** по переменным **x**.

Дадим примеры использования этих команд:

```
> curl(grd,[x,y,z]);
```

$$[\,0,\ 0,\ 0\,]$$

```
> laplacian(f,[x,y,z]);
```

$$4 - \sin(z)$$

```
> jacobian(grd,[x,y,z]);
```

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & -\sin(z) \end{pmatrix}$$

В заключение приведем некоторые команды пакета **linalg** для работы с полиномами и уравнениями:

sylvester(pol1,pol2,x) – вычисление матрицы Сильвестра для полиномов **pol1** и **pol2**, зависящих от переменной **x**;

bezout(pol1,pol2,x) – формирование матрицы Безу двух полиномов;

genmatrix(eqns,vars) – формирование матрицы системы для набора уравнений **eqns** по переменным **vars**.

6. Дифференциальные уравнения

В данной главе речь пойдет о решении дифференциальных уравнений. Вначале будет описана многоцелевая команда **dsolve**, используемая для аналитического и численного решения обыкновенных дифференциальных уравнений. Затем будет описана специальная структура **DESol**, предоставляющая возможность аналитического оперирования с неявно заданным решением дифференциального уравнения. Наконец, будут описаны команды пакета **DEtools**, ориентированного главным образом на численное решение задачи Коши для дифференциальных уравнений.

6.1. Точные и приближенные решения

Для нахождения аналитических решений дифференциальных уравнений как в квадратурах, так и приближенно, а также для численного решения задачи Коши применяется команда **dsolve**, причем для всех случаев используется единый формат команды:

dsolve(eqns,vars,option).

Здесь **eqns** – одно дифференциальное уравнение или система уравнений относительно неизвестных функций **vars**, а **option** – дополнительные условия, позволяющие указать метод решения задачи (например, **type=numeric** – для численного решения).

Если дополнительных условий нет, то Maple пытается найти аналитическое решение задачи, так как по умолчанию принято, что **type=exact**. При этом выдаваемое решение будет содержать неопределенные константы *_C1*, *_C2*, ... (и т.д.), или параметр *_T* в том случае, когда решение выдается в неявном виде. Если требуется решение в явном виде, т.е. для функций **vars**, то необходимо указать дополнительное условие **type=explicit** (или **explicit**).

Для задачи Коши в уравнения **eqns** нужно включить также начальные условия, а для краевой задачи – соответственно краевые условия. Если Maple может найти решение, но число краевых или начальных условий меньше порядка системы, то в ответе будут фигурировать неопределенные константы.

При составлении уравнений для указания производной применяются команды **diff** и **D**, а для обозначения производной в начальных и краевых условиях используется команда **D**.

Рассмотрим пример задания дифференциального уравнения и обращения к команде **dsolve** для его решения. Обратим внимание на то, что при неполном задании уравнения (отсутствует знак равенства и правая часть) система Maple пополняет уравнение нулевой правой (или левой) частью.

```
> deqn:=diff(y(x),x$2)+3*diff(y(x),x)+2*y(x);
```

$$deqn := \left(\frac{\partial^2}{\partial x^2} y(x)\right) + 3\left(\frac{\partial}{\partial x} y(x)\right) + 2\,y(x)$$

```
> dsolve(deqn,y(x));
```

$$y(x) = \_C1\, e^{(-2x)} + \_C2\, e^{(-x)}$$

Решение линейного уравнения второго порядка выдано в виде суммы экспонент с произвольными константами. Заметим, что данное уравнение при помощи оператора **D** может быть записано в следующем виде:

```
> deqn:=(D@@2)(y)(x)+3*D(y)(x)+2*y(x);
```

$$deqn := D^{(2)}(y)(x) + 3\,D(y)(x) + 2\,y(x)$$

Рассмотрим краевую задачу для того же уравнения. Определим краевые условия и воспользуемся командой **dsolve**:

```
> bvp:=y(0)=0,y(1)=1;
```

$$bvp := y(0) = 0,\ y(1) = 1$$

```
> dsolve({deqn,bvp},y(x));
```

$$y(x) = \frac{e^{(-2x)}}{e^{(-2)} - e^{(-1)}} - \frac{e^{(-x)}}{e^{(-2)} - e^{(-1)}}$$

Отметим, что переменная **bvp** имеет тип **exprseq** (последовательность выражений), и объединение ее с уравнением **deqn** потребовало заключения их в фигурные скобки для оформления выражения типа **set** (множество). Если же переменную **bvp** определить как множество, то первый аргумент команды **dsolve** должен быть оформлен как сумма множеств. Для этого нужно взять пе-

ременную **deqn** в фигурные скобки и применить специальную операцию сложения (**union**):

```
> bvp:={y(0)=0,y(1)=1};
> dsolve({deqn} union bvp,y(x));
```

Команда **dsolve** предоставляет возможность определения базисных функций фундаментального решения дифференциального уравнения. Приведем соответствующий пример, заодно напомнив, что Maple легко обрабатывает символьные выражения, где бы они ни встретились: здесь первый параметр команды содержит выражение, задающее новое, чуть измененное уравнение.

```
> dsolve(deqn-a*exp(x),y(x),output=basis);
```

$$\left[\left[e^{(-2x)}, e^{(-x)}\right], \frac{1}{6} a e^{x}\right]$$

Если для дифференциального уравнения не удается факторизовать характеристический полином, то ответ выдается при помощи функции **RootOf**. Например, для дифференциального уравнения четвертого порядка имеем

```
> ode:=diff(y(x),x$4)-diff(y(x),x$3)
>       +a*diff(y(x),x)-a*y(x):
> dsolve(ode,y(x));
```

$$y(x) = \_C2\, e^{x} + \left(\sum_{\_R = \%1} \_C1_{\_R}\, e^{(\_R x)} \right)$$

$$\%1 := \mathrm{RootOf}\left(\_Z^{3} + a\right)$$

Обратим внимание, что поскольку в качестве разделителя взято двоеточие, то отсутствует область вывода для задающей дифференциальное уравнение команды. Здесь и далее в простых случаях команды с опущенным выводом используются для сокращения дублируемой в областях ввода и вывода информации.

Для решения нелинейных задач, которые не поддаются аналитическим методам, могут быть использованы численные процедуры. При этом, однако, должны быть определены все параметры. Когда решения в замкнутой форме нет, а численные подходы невозможны или нежелательны, то могут быть использованы приближенные методы. Для этого команда **dsolve** имеет опцию **series** для поиска

решения в виде разложения в ряд и опцию **laplace** для применения метода преобразования Лапласа.

Приведем примеры обращения к команде **dsolve** с опцией **series**. Вначале переопределим значение переменной **Order**, задающей порядок малости, до которого будет вычисляться разложение, а затем получим решения.

```
> Order:= 4:
> dsolve(diff(y(x),x$2)-y(x)^2=1,y(x),series);
```

$$y(x) = y(0) + D(y)(0)\,x + \left(\frac{1}{2}y(0)^2 + \frac{1}{2}\right)x^2 + \frac{1}{3}y(0)\,D(y)(0)\,x^3 + O(x^4)$$

```
> dsolve({diff(y(x),x)-b*sin(y(x))=x,y(0)=0},
>        y(x),series);
```

$$y(x) = \frac{1}{2}x^2 + \frac{1}{6}b\,x^3 + O(x^4)$$

6.2. Численные решения

Для получения численного решения системы дифференциальных уравнений команда **dsolve** должна быть запущена с опцией **numeric**. В результате будет создана процедура, к которой можно обращаться для вычисления отдельных значений и для построения графика решения на заданном промежутке.

Приведем пример решения задачи Коши для рассмотренного выше линейного уравнения второго порядка

```
> deqn:=(D@@2)(y)(x)+3*D(y)(x)+2*y(x):
> init:=y(0)=0,D(y)(0)=1;
```

$$init := y(0) = 0,\ D(y)(0) = 1$$

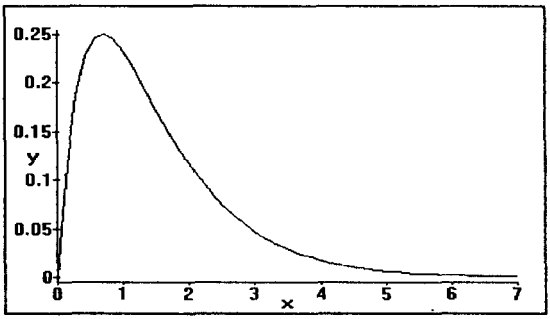
```
> F:=dsolve({deqn,init},y(x),numeric);
```

$$F := proc(rkf45\_x)\ \ldots\ end$$

```
> F(1.5);
```

$$\left[x = 1.5,\ y(x) = .1733430934268153,\ \frac{\partial}{\partial x}y(x) = -.1235560268235766\right]$$

```
> plots[odeplot](F,[x,y(x)],0..7,labels=[x,y]);
```



В приведенном примере для построения графика решения использована команда **odeplot** из пакета **plots**, а кроме того, в Maple имеется инструментальный пакет для представления результатов решения дифференциальных уравнений **DEtools**, о котором речь пойдет ниже.

Если указать опцию **output=listprocedure**, то в результате сформируется список процедур – по процедуре на каждую переменную:

```
> p:=dsolve({diff(x(t),t)=y(t),diff(y(t),t)=-x(t),
>              x(0)=1,y(0)=0},{x(t),y(t)},
>            type=numeric,output=listprocedure);
```

$$p := [t = proc(t) \ldots end,\ y(t) = proc(t) \ldots end,\ x(t) = proc(t) \ldots end]$$

Далее к решению можно обращаться, указывая нужную переменную. Для того чтобы выдать график решения, следует использовать параметрическую форму

```
> X:=subs(p,x(t)): Y:=subs(p,y(t)):
> plot([X,Y,0..2*Pi]);
```

Заданная система представляет собой уравнения математического маятника, так что на графике получится окружность.

При помощи команды **dsolve** можно получить решение в виде таблицы. Для этого следует указать опцию **value=X**, где массив X содержит точки, в которых будет вычислено решение (см. пример в гл. 10 "Мини-исследования").

Остановимся подробнее на применении команды **dsolve** для численного решения дифференциальных уравнений. Возможны два варианта соответствующего указания

```
dsolve(eqns,vars,numeric)
```

и

```
dsolve(eqns,vars,type=numeric,option).
```

Здесь дополнительные параметры (**option**) нужны для задания метода численного решения задачи Коши: **method=rkf45** (метод Рунге–Кутта–Фельберга – РКФ, порядка 4-5) или **method=dverk78** (метод Рунге–Кутта, порядка 7-8) и для определения дополнительных параметров этих методов. По умолчанию используется метод Рунге–Кутта–Фельберга.

Приведем ряд дополнительных параметров для метода РКФ вместе с краткими пояснениями: **'abserr'=aerr** – величина абсолютной погрешности, по умолчанию принято, что **aerr=Float(1,2-Digits)**; **'minrel'=minr** – минимальная величина относительной погрешности (вещественное число), по умолчанию принято, что **aerr=Float(1,1-Digits)**; **'relerr'=rerr** – величина относительной погрешности, по умолчанию принято, что **rerr=Float(1,2-Digits)**. Здесь первым параметром команды **Float** задается мантисса, а вторым – порядок формируемого числа.

6.3. Структура DESol

Для работы с дифференциальными уравнениями в Maple V Release 3 введен специальный объект **DESol** – решение дифференциального уравнения

```
DESol(deqs,vars,inits)
```

Данная структура позволяет работать с решением, не имея его в явном виде. Это своеобразное развитие функции **RootOf** применительно к дифференциальным уравнениям. Объект **DESol** можно интегрировать, дифференцировать, находить для него разложения и, конечно, получать численное решение.

Так, можно проверить, является ли некоторое выражение интегралом данного уравнения.

```
> sol:=DESol(diff(y(x),x,x)+a*y(x),y(x));
```

$$sol := \mathrm{DESol}\left(\left\{\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)+a\,\mathrm{y}(x)\right\},\{\mathrm{y}(x)\}\right)$$

```
> integ:=diff(sol,x)^2+a*sol^2;
```

$$integ := \left(\frac{\partial}{\partial x}\mathrm{DESol}\left(\left\{\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)+a\,\mathrm{y}(x)\right\},\{\mathrm{y}(x)\}\right)\right)^2 + a\,\mathrm{DESol}\left(\left\{\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)+a\,\mathrm{y}(x)\right\},\{\mathrm{y}(x)\}\right)^2$$

```
> diff(integ,x);
```

$$0$$

Следующий пример показывает, что объект DESol может выступать аргументом команды разложения в ряд **series** как обычная функция. Вначале зададим уравнение, начальные условия и определим сам объект

```
> ode:=diff(y(x),x,x)+y(x)^3=sin(x);
> ini:={y(0)=0,D(y)(0)=1};
```

$$ode := \left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)+\mathrm{y}(x)^3=\sin(x)$$

$$ini := \{\mathrm{y}(0)=0, \mathrm{D}(y)(0)=1\}$$

```
> dde:=DESol(ode,y(x),ini);
```

$$dde := \mathrm{DESol}\left(\left\{\left(\frac{\partial^2}{\partial x^2}\mathrm{y}(x)\right)+\mathrm{y}(x)^3-\sin(x)\right\},\{\mathrm{y}(x)\},\{\mathrm{y}(0)=0,\mathrm{D}(y)(0)=1\}\right)$$

Несложно убедиться, что запуск команды

```
> dsolve({ode} union ini, y(x));
```

оставит область вывода пустой – Maple не может найти аналитическое решение данной задачи Коши. Остается попробовать численное решение и разложение в ряд. Следующая сложносочиненная команда, во-первых, вычисляет несколько членов ряда (команда **series**), а во-вторых, преобразует полученное в полином при помощи команды **convert**

```
> poly:=convert(series(dde,x=0),polynom);
```

$$poly := x + \frac{1}{6}x^3 - \frac{7}{120}x^5$$

Затем дается команда о численном решении задачи, и генерируется процедура, обращаясь к которой, можно это решение получать. Обратим внимание на то, как задается задача Коши: уравнение **ode** при помощи фигурных скобок превращается в множество и объединяется с множеством начальных условий **ini**.

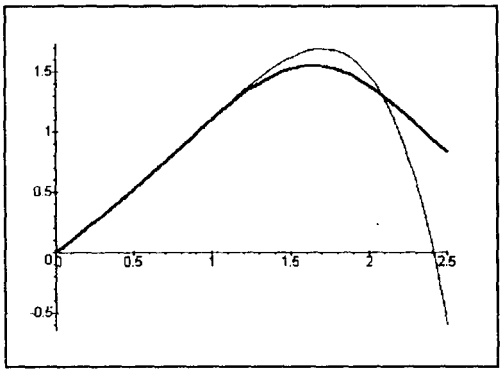
```
> f:= dsolve({ode} union ini, y(x),type=numeric);
```

$$f := proc(rkf45\_x) \ldots end$$

Наконец, подключается пакет графики **plots**, при помощи разных графических команд строятся графики в едином масштабе и сводятся воедино при помощи команды **display**.

```
> with(plots):
> d_f:=odeplot(f,[x,y(x)],0..2.5,thickness=2):
> d_p:=plot(poly,x=0..2.5): display({d_p,d_f});
```

В результате получается график с двумя кривыми, где численное решение задачи представлено более жирной кривой.



6.4. Пакет DEtools

Для численного решения задачи Коши, построения фазовых портретов и векторных полей имеется пакет **DEtools**, состоящий из семи команд:

DEplot – рисование графиков решений дифференциального уравнения или системы уравнений;

DEplot1 – рисование графиков решений дифференциального уравнения первого порядка;
DEplot2 – рисование графиков решения дифференциального уравнения второго порядка;
Dchangevar – замена переменных в дифференциальном уравнении;
PDEplot – рисование трехмерного графика решения квазилинейного уравнения первого порядка в частных производных;
dfieldplot – рисование двумерного поля направлений (векторного поля);
phaseportrait – рисование фазового портрета для системы двух уравнений первого порядка.

Для использования команд из пакета **DEtools** нужно либо подключить весь пакет при помощи команды

```
with(DEtools),
```

либо обращаться к нужной команде **command** при помощи вызова

```
with(DEtools,command).
```

Если имя **command** конфликтует с именем другой команды, то возможно использование следующей конструкции:

```
readlib(DEtools)[command].
```

Начнем описание пакета **DEtools** с команды **DEplot**, предназначенной для вывода графиков решений системы дифференциальных уравнений. Возможные форматы обращения к команде имеют вид

```
DEplot(deqn,vars,trange,inits,options)
```

и

```
DEplot(deqn,vars,trange,inits,xrange,yrange,options)
```

Здесь приняты следующие обозначения для параметров: **deqn** – система **n** уравнений первого порядка или одно уравнение n-го порядка; **vars** – имена переменных; **trange** – область изменения независимой переменной; **inits** – начальные условия; **xrange** – масштаб для независимой переменной; **yrange** – масштаб для переменных решения; **options** – дополнительные условия (могут отсутствовать).

При задании уравнений не должны использоваться никакие символические переменные. Первый аргумент **deqn** может принимать разные формы. Например, для системы из двух уравнений первого порядка допустимы следующие варианты:

```
  [diff(x(t),t)=f1(t,x,y), diff(y(t),t)=f2(t,x,y)],
[diff(x(t),t)-f1(t,x,y)=0, diff(y(t),t)-f2(t,x,y)=0],
  [diff(x(t),t)-f1(t,x,y), diff(y(t),t)-f2(t,x,y)],
```

или

```
[D(x)(t)=f1(t,x,y), D(y)(t)=f2(t,x,y)].
```

Здесь при помощи `f1(t,x,y), f2(t,x,y)` обозначены правые части дифференциальных уравнений, в которых могут присутствовать независимая переменная (`t`), искомые функции (`x, y`) и константы. Для уравнений высоких порядков важна выделенность старшей производной

```
[diff(x(t),t$n)=f(t,x)],
[diff(x(t),t$n)-f(t,x)=0].
```

Здесь функция `f(t,x)` может включать также производные от переменной `x`; `n` – целое число, и знаком ($) отмечается производная порядка `n`.

Аргумент **vars** перечисляет имена переменных. Для неавтономной системы уравнений возможны две формы задания **vars**: `[t,x,y,...]` или `[x(t),y(t),...]`. Для автономных уравнений следует использовать запись `[x,y]`, если необходимо изображение фазового портрета. Заметим также, что это ускоряет вычисление поля направлений для автономных систем.

Аргумент **trange** определяет интервал изменения независимой переменной и может задаваться в виде `a..b` или `t=a..b`, причем `a` и `b` должны быть вещественными константами.

Аргумент **inits** задает набор начальных условий в одной из следующих форм:

```
{[t0,x0,y0],[t1,x1,y1], ...}
```

или

```
{[x(t0)=x0,y(t0)=y0],[x(t1)=x1,y(t1)=y1],...}
```

Видно, что стартовые значения независимой переменной (`t`) для разных начальных точек могут не совпадать, решение для переменных будет получено для всего интервала **trange**, а

задание начального значения специфицирует интегральную кривую.

Для уравнения n-го порядка относительно функции **x(t)** вначале задается значение независимой переменной, а далее – значения функции и производных: **[t0,x0,x'0,x''0,...]**.

Если число начальных условий меньше числа переменных, то недостающие значения будут выбраны нулевыми и будут выданы предупреждающие сообщения. Если начальные условия не заданы, то будет выдано только поле направлений. При этом должны быть сделаны все назначения для изображения этого поля. Поле направлений не выдается для трехмерных картин.

Информацию о всех опциях команды **DEplot** можно получить, обратившись к справке:

```
?DEtools[options]
```

По умолчанию для неавтономных уравнений выдаются кривые решений в трехмерном пространстве первых трех переменных **vars**. Заданием параметра сцены (**scene=...**) можно получить двумерные проекции или графики для различных переменных. В случае системы двух автономных уравнений по умолчанию выдается фазовый портрет для исходных переменных.

В командах пакета **DEtools** можно использовать опции, которые имеются в пакетах двумерной и трехмерной графики. Так, например, для задания заголовка можно применять конструкцию **title=string**. Описание предоставляемых при этом возможностей и примеры будут даны в гл. 9 "Графика в Maple".

Опишем опции пакета **DEtools**.

x=c1..d1, y=c2..d2 – границы вывода для переменных **x** и **y** указываются при помощи вещественных констант.

stepsize=h – задание шага интегрирования. По умолчанию берется **(b-a)/20**, где **[a,b]** – отрезок интегрирования. Поскольку Maple не является специализированной программой для решения систем дифференциальных уравнений, то нужно иметь в виду, что уменьшение шага дает не только повышение точности и гладкости получаемых кривых, но и увеличение времени расчета. Так как все выводимые на график точки запасаются в памяти, то вполне возможна ситуация ее исчерпания.

`iterations=n` – число шагов интегрирования между выводимыми точками. По умолчанию оно равно единице. Эта опция полезна для повышения точности расчета, поскольку те точки, которые не выводятся на график, не накапливаются в памяти.

`arrows=type` – тип стрелок при изображении поля направлений, здесь `type` – один из следующих терминов: `NONE` (по умолчанию), `THIN`, `SLIM`, `THICK`, `LINE`.

`grid=[n,m]` – сетка для рисования поля направлений, по умолчанию принято `[10,10]`.

`limitrange=true` – опция, позволяющая прекратить расчет, если интегральная кривая вышла за границы вывода для переменной `x` или `y`. По умолчанию задано `limitrange = false`. Если границы для x и y не определены, то расчет ведется независимо от заданной опции.

Назначение следующей опции – задавать вид рисунка, например `scene=[x,y]` означает изображение двумерного фазового портрета (график функции `y(x)`), а `scene=[t,x,y]` и `scene=[x,y,t]` означают трехмерные картины с различным расположением осей.

`arraysize=n` – определение размера рабочего массива для хранения результатов интегрирования.

`method=scheme` – задание метода интегрирования; здесь в качестве `scheme` выступает название одного из методов: `'euler'`, `'backeuler'`, `'impeuler'` или `'rkf45'`. Заметим, что для предупреждения конфликта с другими терминами, использующими имя Эйлера, название `'euler'` следует приводить в кавычках.

Отметим, что можно написать собственную процедуру интегрирования, указав опцию `method=My_Scheme`. Заголовок процедуры при этом должен иметь следующий вид: `My_Scheme:=proc(ndvar,h,endpt,iter,pt,F)`, `ndvar` – число переменных, `h` – шаг интегрирования, `endpt` – длина отрезка интегрирования, `iter` – число шагов между точками, запасаемыми в массиве `pt` для вывода на график (каждая точка – это тройка `[t,x,y]`), `F` – функция, вычисляющая правые части системы дифференциальных уравнений.

Приведем примеры использования команды **DEplot** для расчета уравнения Ван дер Поля с вынуждающей гармонической силой. Так как решается неавтономное уравнение, то по умолчанию выдается траектория в пространстве **(t,x.y)**, а при переназначении параметра сцены – фазовый портрет.

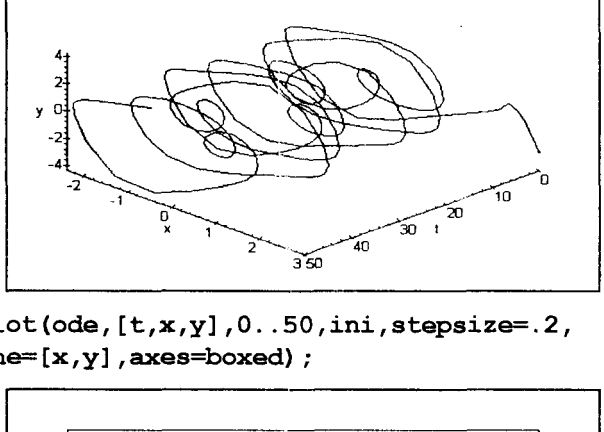
```
> with(DEtools):
> ode:=[y,-x+y*(1-x^2)+2.5*sin(1.7*t)];
```

$$ode := \left[y, -x + y\left(1 - x^2\right) + 2.5\sin(1.7\,t)\right]$$

```
> ini:={[0,3,-3]};
```

$$ini := \{[0, 3, -3]\}$$

```
> DEplot(ode,[t,x,y],0..50,ini,stepsize=.2);
```




```
> DEplot(ode,[t,x,y],0..50,ini,stepsize=.2,
> scene=[x,y],axes=boxed);
```

Команды **DEplot1** и **DEplot2** имеют тот же формат, что и рассмотренная команда **DEplot**, но с поправкой на заданную номером размерность решаемых уравнений. При решении задач малых размерностей команда **DEplot** обращается к этим командам.

Для системы двух уравнений первого порядка, разрешенных относительно производных, можно не вводить сами уравнения, а задать их правые части. Это может быть сделано при помощи одного из следующих способов:

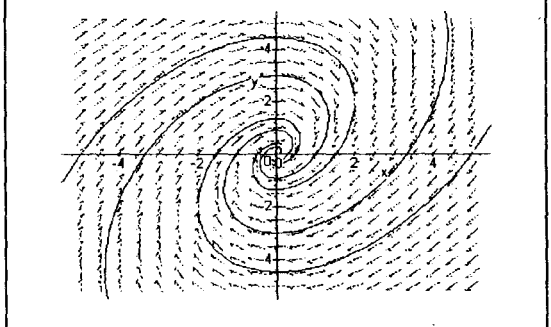
```
[f1(t,x,y), f2(t,x,y)],
[(t,x,y)->f1(t,x,y), (t,x,y)->f2(t,x,y)],
(t,x,y)->[f1(t,x,y), f2(t,x,y)].
```

Для автономных уравнений идентификатор независимой переменной (например, **t**) может быть опущен:

```
[(x,y)->x+y, (x,y)->x-y].
```

Важно, чтобы число аргументов функции соответствовало числу переменных, определенному заданием параметра **vars**. Наконец, систему двух линейных уравнений также можно задать, приведя числовую матрицу коэффициентов для правой части уравнений. Например, построение поля направлений и нескольких траекторий, ведущих к устойчивому фокусу, легко реализуется при помощи следующих команд:

```
> A:= linalg[matrix](2,2,[-1,1,-2,0]):
> readlib(DEtools)[DEplot2](A,[x,y],-2..6,
>  {[0,0,3],[0,-5,0], [0,0,-3],[0,5,0]},
>  stepsize=.1,scene=[x,y], x=-5..5, y=-5..5);
```



Обратим внимание на то, что графики **y(x)** выданы для всего интервала изменения независимой переменной **t** ([-2..6]), хотя начальные условия заданы в точке **t=0**. Здесь начальные условия выделяют те интегральные кривые, которые строит Maple.

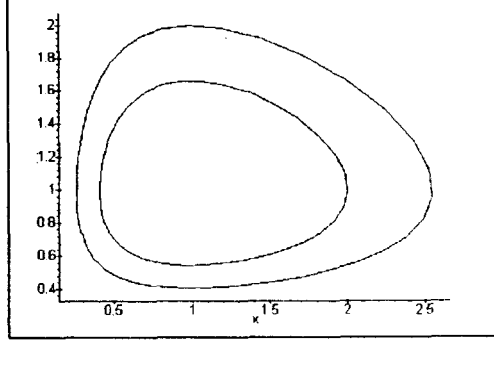
Приведенные возможности являются довольно специальными, но, развиваясь от простого к менее простому, система Maple обретает более высокую математическую квалификацию, и, следовательно, можно ожидать распространения этих умений на системы более высоких размерностей.

Команда **phaseportrait** предназначена для изображения фазовых кривых на плоскости и обращается к команде **DEplot1**, если задано одно уравнение, и к **DEplot2** для системы двух уравнений первого порядка. Эта команда имеет следующий формат:

```
phaseportrait(deqn,vars,trange,inits,options)
```

Здесь по-прежнему **deqn** – дифференциальное уравнение или система двух уравнений первого порядка, **vars** – имена переменных в квадратных скобках, **trange** – интервал изменения независимой переменной, **inits** – множество начальных условий, **options** – дополнительные параметры, частично описанные выше для команды **DEplot**. При задании уравнений здесь достаточно указать их правые части. Приведем пример построения фазовых кривых для системы уравнений Лотки–Вольтерра.

```
> with(DEtools): f:=[2*x*(1-y),y*(x-1)]:
> phaseportrait(f,[x,y],0..5,{[0,2,1],[0,1,2]},
>     stepsize=.1,title=`Lotka-Volterra equation`);
```



Команда **dfieldplot** предназначена для изображения векторного поля (поля направлений) и согласована по параметрам с командой **DEplot**, за исключением начальных условий inits, задания которых не нужно. Можно сказать, что команда **DEplot2** есть объединение команд **dfieldplot** и **phaseportrait**.

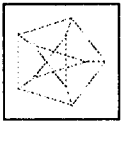
Команда **Dchangevar(expr,deqn,cns)** позволяет проводить замену переменных в дифференциальном уравнении. Здесь **expr** – набор соотношений, задающих замену переменных, **deqn** – дифференциальное уравнение, **cns** – дополнительный список имен переменных, которые считаются константами при замене переменных. Данная команда необходима, поскольку команда подстановки **subs** не позволяет проводить замену в выражении производной. Приводимый пример замены переменных для уравнения гармонического осциллятора иллюстрирует работу команды.

```
> with(DEtools): de:=diff(x(t),t,t)/a^2+x(t)=0;
```

$$de := \frac{\frac{\partial^2}{\partial t^2} x(t)}{a^2} + x(t) = 0$$

```
> Dchangevar({t = tau/a,x(t) = y(tau)},de,{a});
```

$$\left(\frac{\partial^2}{\partial \tau^2} y(\tau)\right) + y(\tau) = 0$$



7. Математические библиотеки

О возможностях пакетов аналитических (символьных) вычислений судят по их умению решать самые разнообразные математические задачи. Maple оснащен большим количеством команд, организованных в библиотеки (пакеты), специализированные по темам.

В предыдущих главах были рассмотрены команды и библиотеки, без которых, по нашему мнению, невозможно содержательное введение в пакет Maple. Подробное изложение в рамках данной книги всех команд стандартной библиотеки и наполнения других библиотек не представляется возможным. Поэтому ограничимся описанием некоторых команд стандартной библиотеки и эскизными характеристиками большинства библиотек. Напомним, что для обращения к командам пакета с именем **package** его нужно подгрузить при помощи команды

with(package).

7.1. Определение абстрактных операторов

В Maple существует возможность работы с абстрактными, не заданными явно математическими операторами. Для определения оператора и его свойств используется команда **define**:

define(Linear(oper)) – для определения линейного оператора **oper**;

define(Group(oper,Identity,Inverse)) – для задания группы. Здесь **oper** – имя оператора, **Identity** – нулевой элемент, **Inverse** – оператор, вычисляющий обратный.

Приведем примеры использования этих конструкций.

```
> define(Linear(A));
> A(x*sin(x)+5*y*v);
```

$$A(x\sin(x)) + 5\,A(y\,v)$$

```
> define(Group(G,II,F));
```

```
> expand(F(G(a,b,c)));
```

$$G(F(c), F(b), F(a))$$

```
> G(a,F(G(a)));
```

$$II$$

Команда **define** имеет также следующий формат:

define(oper,property1,property2, ...).

Здесь **oper** – оператор, а в качестве задающих свойства оператора опций **property1**, **property2**,... могут выступать зарезервированные служебные слова, уравнения и соотношения вида $f(x) = value$.

Перечислим служебные слова:

unary – оператор действует на один аргумент;

binary – оператор действует на два аргумента;

associative – оператор является ассоциативным, т.е. $f(x,f(y,z)) = f(f(x,y),z) = f(x,y,z)$;

commutative или **symmetric** – оператор симметричен, т.е. $f(x,y) = f(y,x)$;

antisymmetric – оператор кососимметричен , т.е. $f(x,y) = -f(y,x)$;

inverse=g – определяет оператор **g**, обратный к определяемому;

identity=x – определяет нуль для оператора;

zero=expr – задает выражение **expr** для оператора со свойством: $f(expr)=expr$.

Поясним сказанное примерами:

```
> define(f,associative,antisymmetric,zero=x^2);
> f(x,z,y)+f(z,x,y);
```

$$0$$

```
> f(x^2,x^2,z,y);
```

$$f(0, z, y)$$

```
> define(`&v`,associative,commutative,
>         inverse=g,identity=II);
> x &v (y &v z) &v (II &v x);
```

$$\&v(y, x, x, z)$$

```
> x &v g(y) &v y;
```

$$x$$

Для задания некоторых свойств оператора используются уравнения. Чтобы свойство выполнялось для всех аргументов или для аргументов из некоторого класса, используется описатель `forall`. Использование уравнений для определения свойств оператора и описателя `forall` иллюстрирует следующий пример:

```
> define(F,unary,F(2)=25,
>        forall(integer(x),F(x)=x-1));
> F(235); F(2); F(1/34);
```

$$234$$

$$25$$

$$F\left(\frac{1}{34}\right)$$

7.2. Интегральные преобразования

Ряд важных команд стандартной библиотеки предназначен для проведения интегральных преобразований Фурье, Лапласа, Меллина и др.

Например, для вычисления преобразования Фурье

$$F(w)=\int_{-\infty}^{\infty} f(t)\exp(-itw)dt$$

применяется следующая команда:

```
fourier(f,t,w),
```

которую перед использованием нужно вызвать командой `readlib(fourier)`. Выражение `f` может включать экспоненты, полиномы, тригонометрические функции, дельта-функцию Дирака (`Dirac(t)`), функцию Хевисайда (`Heaviside(t)`), а также производные и интегралы. Можно задавать также нестандартные фурье-преобразования.

Далее перечислим в алфавитном порядке другие команды.

`FFT(n,aRe,aIm)` — вычисление быстрого преобразования Фурье, `aRe` и `aIm` – массивы действительных и мнимых частей размерности 2^n. Нужен предварительный вызов при помощи команды `readlib(FFT)`.

iFFT(n,aRe,aIm) — обратное быстрое преобразование Фурье; **aRe** и **aIm** – массивы действительных и мнимых частей размерности 2^n.

invfourier(exp,var,name) — обратное преобразование Фурье для выражения **expr** по переменной **var** и с новой переменной **name**.

invlaplace(exp,var,name) — обратное преобразование Лапласа выражения **expr** по переменной **var** и с новой переменной **name**.

mellin(exp,var,name) — преобразование Меллина выражения **expr** по переменной **var** и с новой переменной **name**.

Приведем пример обращения к процедуре быстрого преобразования Фурье

```
> readlib(FFT); x := array([7.,5.,6.,9.]):
> y := array([0,0,0,0]): FFT(2,x,y);
                                4
> print(x);
                        [ 27., 1., -1., 1. ]
> print(y);
                         [ 0, 4., 0, -4. ]
```

7.3. Интерполяция

В Maple имеется несколько команд, реализующих обычную и сплайн-интерполяцию. Массивы, задающие узлы интерполяции, могут быть не упорядочены, но не должны содержать одинаковых элементов.

Для построения интерполяционного многочлена относительно переменной **var** по таблице, заданной векторами **X,Y**, используется команда **interp(X,Y,var)**.

Построение сплайна относительно переменной **var** по таблице, заданной векторами **X,Y**, производится при помощи команды **spline(X,Y,var,d)**. Здесь параметр **d** определяет порядок сплайна, который может быть линейным (**linear**), квадратичным (**quadratic**), кубическим (**cubic**) и четвертой степени (**quartic**). По умолчанию строится сплайн третьего порядка. Перед использованием команды требуется ее вызов: **readlib(spline)**. Результатом действия команды будет построение сплайна в виде условного

выражения, например присвоенного имени **rez**. Для оформления сплайна в виде процедуры нужно использовать следующую конструкцию:

```
s:=`spline/makeproc`(rez,x).
```

Здесь **s** – имя формируемой процедуры, **x** – независимая переменная. Пример применения сплайн-интерполяции дан в гл. 10 "Мини-исследования".

7.4. Ортогональные полиномы

Пакет **orhtopoly** умеет оперировать со следующими видами ортогональных полиномов:

G(n,a,x) – **n**-й полином Гегенбауэра (семейство ультрасферических полиномов).
H(n,x) – **n**-й многочлен Эрмита.
L(n,x) – **n**-й многочлен Лагерра.
L(n,a,x) – **n**-й обобщенный многочлен Лагерра.
P(n,x) – **n**-й многочлен Лежандра.
P(n,a,b,x) – **n**-й многочлен Якоби.
T(n,x) – **n**-й обобщенный многочлен Чебышева.
U(n,x) – **n**-й многочлен Чебышева второго рода.

Более детальное описание каждого полинома может быть получено при помощи команды справки

```
?orthopoly[letter],
```

где **letter** означает начальную букву соответствующего полинома.

7.5. Теория чисел

Пакет Maple имеет много команд для эффективной работы в теории чисел. Некоторые из этих команд находятся в стандартной библиотеке, но большинство содержится в трех библиотеках: **numtheory**, **GaussIn** и **padic**. В основном и входными параметрами, и результатами действия этих команд выступают числа – целые, рациональные, цепные дроби, гауссовы целые (комплексные целые), p-адические, простые, а также несколько известных иррациональных чисел.

Для выделения числителя и знаменателя рациональной дроби можно использовать команды стандартной библиотеки **numer** и **denom**. Если целое число (знаменатель, числитель) не помещается в строке вывода, то оно переносится на следующую строку, а в качестве знака переноса ставится обратный слэш (\). Maple предоставляет возможность работы с целыми числами, у которых более пятисот тысяч (500000) цифр, но при операциях с такими числами на персональном компьютере, скорее всего, возникнут проблемы с машинным временем и памятью.

Приведем примеры некоторых возможностей данных пакетов. Для решения полиномиальных уравнений с рациональными коэффициентами можно использовать команду **cfracpol** из библиотеки **numtheory**. В этом случае ответ выдается при помощи цепных дробей; команда **cfrac** служит для преобразования обычной дроби в цепную и наоборот.

```
> with(numtheory):
> a:=cfracpol(68*x^4+364*x^3-281*x^2-
>             91*x+66,20);
```

$$a := [-1, 2], [0, 2], [0, 1, 1, 1, 5], [-6]$$

```
> b:=cfrac(a[3]); cfrac(b);
```

$$b := \frac{11}{17}$$

$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{5}}}}$$

Для демонстрации возможностей библиотеки **numtheory** при поиске простых чисел воспользуемся следующим тривиальным примером: число проверяется на простоту и ищется следующее за ним простое число.

```
> d:=7654321: isprime(d); nextprime(d);
```

false

7654337

Поиск ближайшего гауссова числа и проверку суммы этого числа и мнимой добавки на простоту иллюстрирует следующий пример:

```
> with(GaussInt):
> g:=GInearest(sqrt(3+I*5)); GIprime(g+I*4);
```

$$g := 2 + I$$

true

Для p-адических чисел имеется собственное довольно специфическое представление; для преобразования в данные других типов нужно использовать команду **op**, которая выдает внутреннюю структуру числа и предоставляет доступ к его операндам.

Приведем пример использования p-адических чисел для действий с экспонентой и логарифмом.

```
> with(padic):
> Digitsp:=6; evalp(exp(5),5); logp (");
```

$$Digitsp := 6$$

$$1 + 5 + 3\,5^2 + 3\,5^3 + 4\,5^4 + O(5^5)$$

$$5 + O(5^6)$$

7.6. Статистика

Статистика имеет свою, развитую систему пакетов для обслуживания прикладных задач. Команды Maple для статистических работ предназначены тем категориям пользователей, которые нуждаются в среде, позволяющей легко переходить от одной математической специализации к другой, не расходуя лишнего времени на трансформацию данных и освоение различных программных средств.

Пакет **stats** предоставляет хороший набор команд для анализа данных с вычислением различных средних и квантилей, графического представления данных в виде гистограмм и графиков рассеяния, а также для обработки данных.

Пакет состоит из множества команд, объединенных в подбиблиотеки анализа данных (**describe**), сглаживания (**fit**), преобразования данных (**transform**), генерации случайных чисел согласно заданным распределениям (**random**), численной оценки статистических распределений (**statevalf**) и графики (**statplots**), плюс команда считывания данных из файла **importdata** .

Вследствие такой организации пакета для вызова конкретной команды **command** из подбиблиотеки **subpackage**, особенно в случае конфликта используемых в сессии имен, приходится применять один из следующих форматов:

```
subpackage[command](args),
```

если предварительно загружен весь пакет **stats**, и

```
stats[subpackage,command](args),
```

если пакет **stats** не загружен.

Нужную подбиблиотеку можно загрузить при помощи команды

```
with(stats,subpackage).
```

Для получения информации о командах подбиблиотеки **subpackage** следует воспользоваться справкой

```
?stats[subpackage]
```

или посмотреть соответствующую тему в Help Browser. Там же можно получить сведения о представлении данных (**data**) и об имеющихся статистических распределениях (**distributions**). Наконец, всегда полезно справиться об изменениях в системе (**updates**).

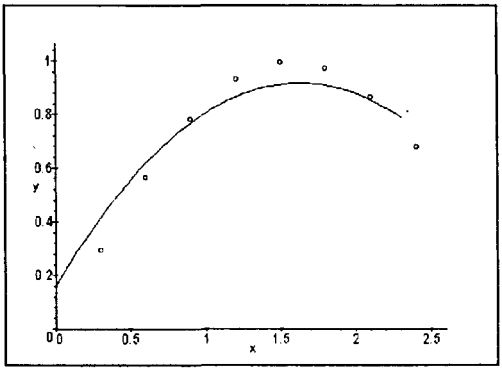
Приведем пример использования команд пакета **stats** для аппроксимации функции синуса квадратичным полиномом. Вначале подгружается сам пакет, устанавливается разрядность, с которой будут проводиться вычисления, и подготавливаются два набора данных – абсциссы и ординаты. Затем методом наименьших квадратов находятся коэффициенты заданного полинома, подгружается графическая подбиблиотека и при помощи команды **display** из графического пакета **plots** на одном рисунке изображаются исходные точки (команда **scatter2d**) и график квадратичного полинома (команда **plot**). Здесь обратим внимание на оформление вызова нужных команд: для запуска команды **leastsquare** применен вызов с префиксом подбиблиотеки **fit**, а перед использованием команды **scatter2d** подгружена графическая подбиблиотека **statplots**.

```
> with(stats): Digits:=3:
> xx:=[evalf(seq(.3*j,j=1..8))]:
> yy:=[evalf(seq(sin(.3*j),j=1..8))]:
```

```
> f:=fit[leastsquare[[x,y],y=a*x^2+b*x+c]]
>                          ([xx,yy]);
```

$$f := y = -.4425\,x^2 + 1.385\,x - .0933$$

```
> with(stats[statplots]):
> plots[display]({plot(rhs(f),x=0..2.3,y=0..1),
>                 scatter2d(xx,yy)},symbol=circle);
```



7.7. Степенные разложения

Пакет **powseries** содержит команды для работы с формальными степенными разложениями. В пакете около двадцати команд, которые позволяют задавать разложения как посредством рекуррентных формул (**powcreate**), так и через обычные полиномы (**powpoly**), а также проводить с ними различные операции. Для хранения разложений используется специальный формат, поэтому для просмотра их нужно переводить в обычные ряды при помощи команды (**tpsform**).

Коротко опишем имеющиеся в пакете команды. Задаваемые разложения получают имена, которые далее представляют разложения в аргументах следующих команд: вычисления суммы (**add**) и разности (**subtract**), умножения на заданное выражение (**multconst**), произведения (**multiply**) и деления (**quotient**), дифференцирования (**powdiff**) и интегрирования (**powint**), вычисления экспоненты (**powexp**) и натурального логарифма (**powlog**), вычисления мультипликативного (**inverse**) и аддитивного (**negative**) обратных, композиции (**compose**) и обратного к композиции разложения (**reversion**).

Имеется также аналог команды **eval** – команда **evalpow**, позволяющая вычислять сложные выражения, в которых участвуют разложения и используются обычные знаки для сложения, вычитания и умножения, а также специальные операции (**powexp**, **powinv**, **powlog**, **powneg**, **powrev**, **powdiff**, **powint**, **powquo**). Для решения линейного дифференциального уравнения в виде формального разложения применяется команда **powsolve**.

Продемонстрируем работу пакета на примере построения формального разложения для решения дифференциального уравнения. Обратим внимание, что при постановке начальных условий отсутствовало условие для третьей производной, и в ответе возникла константа *C3*.

```
> with(powseries):
> eq:=diff(y(x,x$5)=-y(x,y(0)=0,
>          D(y(0)=-1,D(D(y))(0)=-2;
```

$$eq := \frac{\partial^5}{\partial x^5} y(x) = -y(x), y(0) = 0, D(y)(0) = -1, D^{(2)}(y)(0) = -2$$

```
> v:=powsolve({eq,D(D(D(D(y))))(0)=4}):
> tpsform(v,x,5);
```

$$-x - x^2 + \frac{1}{6} C3\, x^3 + \frac{1}{6} x^4 + O(x^5)$$

7.8. Линейная оптимизация

Пакет **simplex** содержит команды для решения задач линейной оптимизации при помощи симплекс-метода. Перед обращением к командам пакета его нужно подгрузить или использовать вызов команды с префиксом пакета.

Для определения максимума линейной функции **f** при ограничениях **h** применяется команда **maximize(f,h)**. Для поиска минимума используется команда **minimize**, а команда **feasible** предназначена для выяснения вопроса, существует ли решение для данной системы ограничений. Другие команды позволяют выполнять операции, реализующие отдельные шаги симплекс-метода. Перечислим их.

setup – задание системы линейных уравнений для последующего определения базиса (перечня переменных) при помощи

команды **basis**; уравнения задаются так, что каждая из переменных базиса входит в левую часть одного уравнения;

convexhull – вычисление выпуклой оболочки для набора точек;

cterm – определение констант для системы уравнений или неравенств;

define_zero – определение наименьшего ненулевого значения (по умолчанию связано с константой **Digits**);

display – выдача заданных линейных уравнений и неравенств в матричной форме;

dual – выдача сопряженной задачи;

pivot – конструирование новой системы уравнений с заданным главным элементом;

pivoteqn – выдача подсистемы для заданного главного элемента;

pivotvar – выдача переменных, имеющих положительные коэффициенты в выражении целевой функции;

ratio – выдача отношений для определения наиболее жесткого ограничения;

standardize – приведение заданной системы уравнений и неравенств к стандартной (в виде неравенств) форме.

Приведем простой пример.

```
> with(simplex): obj:=-x+2*y+3*z:
> cns:={x+2*y-3*z<=4,5*x-6*y+7*z<=8,9*x+10*z<=11}:
> maximize(obj,cns union {x>=0,y>=0});
```

$$\left\{y=\frac{73}{20}, z=\frac{11}{10}, x=0\right\}$$

7.9. Математическая логика

Для использования команды пакета **logic** нужно загрузить либо весь пакет при помощи команды **with(logic)**, либо нужную команду – **with(logic,command)**. Также допускается непосредственное обращение к команде с указанием префикса пакета **logic[command](arg)**, где **arg** – аргументы команды.

Пакет математической логики **logic** состоит из нескольких команд: **bequal** – проверка эквивалентности двух логических выражений, **bsimp** – упрощение булевского выражения, **canon** – представление выражения в каноническом виде, **distrib** –

представление в виде суммы произведений (дизъюнкция, конъюнкция), **dual** – вычисление двойственного данному выражения, **environ** – задание уровня автоматического упрощения выражений (0 – без упрощения), **randbool** – построение случайного булевского выражения для заданных логических переменных, **satisfy** – подбор значений переменных, удовлетворяющих истинности заданного выражения, **tautology** – проверка на тождественную истинность. Наконец, имеется несколько вариантов команды **convert** для преобразования логических выражений к общесистемным.

При построении логических выражений допустимы следующие булевские операции: **&and** (и), **&or** (или), **&not** (отрицание), **&iff** (если и только если), **&nor** (отрицание или), **&nand** (отрицание и), **&xor** (исключающее или) и **&implies** (импликация).

Проиллюстрируем работу пакета на примере построения и упрощения случайного выражения для двух булевских переменных.

```
> with(logic):
> q:=randbool({a,b},CNF); bsimp(q);
```

$$q := (\&\text{not}(a)\ \&\text{or}\ \&\text{not}(b))\ \&\text{and}\ (a\ \&\text{or}\ \&\text{not}(b))\ \&\text{not}(b)$$

7.10. Теория графов

Для работы с графами предназначен пакет **Networks**. Граф задается при помощи команд **new**, **complete**, **cycle** или **petersen** и состоит из вершин и ребер (простых, кратных и петель). Граф представляется в виде процедуры типа **GRAPH**, тело которой обычно не выводится, так как по умолчанию задан режим подавления вывода: **interface(verboseproc=0)**.

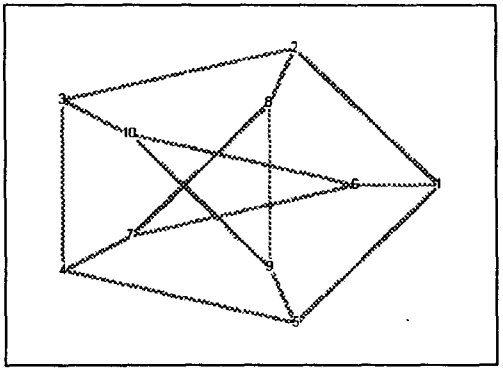
Для создания копии графа имеется команда **duplicate**, причем последующие модификации исходного графа не затрагивают копии. Созданный граф может быть изменен при помощи различных команд: для добавления вершин и ребер служат команды **addvertex** и **addedge** соответственно, для удаления – **delete**. По умолчанию вес вершины принят равным нулю, а имена вершин задаются числами, но могут иметь любые допускаемые Maple имена. Имена ребер по умолчанию даются в формате **e.(1..m)** (т.е. **e1**, **e2** и т.д.),

также допускаются любые, начинающиеся с буквы `e` имена. По умолчанию вес ребра равен единице, но может быть Maple-выражением.

Кроме того, существует много команд, которые реализуют основные операции работы с графами: вычисление потоков в сетях, определение связности, поиск покрывающих деревьев, расчет всех кратчайших путей и т.д.

Для генерации случайного графа служит команда **random**. Для определения характеристик графа предназначены команды **edges** и **vertices**. Приведем простой пример. Определим граф Петерсена и нарисуем его. Затем удалим несколько ребер и вершин и выдадим характеристики модифицированного графа. В заключение посчитаем связность и число возможных разрезов.

```
> with(networks): G:=petersen(): draw(G);
```




```
> delete({e.(1,4,7,12)},G): delete({2,10},G):
> vertices(G); edges(G); ends(G);
```

$$\{7, 8, 9, 1, 3, 4, 5, 6\}$$

$$\{e3, e14, e15, e11, e8, e6, e5\}$$

$$\{\{1, 6\}, \{7, 4\}, \{9, 5\}, \{1, 5\}, \{3, 4\}, \{7, 6\}, \{8, 9\}\}$$

```
> connectivity(G); countcuts(G);
```

$$1$$

$$7$$

7.11. Аппроксимация функций

В пакете **numapprox** собраны команды для приближения функций рациональными полиномами. Перечислим их:

chebpade – вычисление аппроксимации и паде-аппроксимации при помощи полиномов Чебышева,

confracform – преобразование рациональной функции в цепную дробь,

hornerform – преобразование полинома по схеме Горнера,

infnorm – вычисление нормы L_∞,

laurent – разложение функции в ряд Лорана,

minimax – вычисление наилучшей минимаксной рациональной аппроксимации,

pade – вычисление паде-аппроксимации,

remez – вычисление наилучшей минимаксной рациональной аппроксимации по алгоритму Ремеза.

Приведем примеры обращения к командам пакета, выбрав синус в качестве аппроксимируемой и оцениваемой функции. Вначале получим обычное разложение в ряд Лорана и превратим его в рациональную дробь.

```
> with(numapprox):
> c:=laurent(sin(x),x); convert(c,ratpoly);
```

$$c := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^7)$$

$$\frac{-\frac{7}{60}x^3 + x}{1 + \frac{1}{20}x^2}$$

Затем вычислим паде-аппроксимацию и преобразуем полученное в цепную дробь.

```
> p:=pade(sin(x),x,[3,3]); confracform(p);
```

$$p := \frac{-\frac{7}{60}x^3 + x}{1 + \frac{1}{20}x^2}$$

$$-\frac{7}{3}x + \frac{200}{3}\cfrac{1}{x + 20\cfrac{1}{x}}$$

Вычислим норму L_∞ для синуса на заданном отрезке и найдем точку, в которой достигается максимум.

```
> infnorm(sin,-2..2, 'xmax'); xmax;
```

$$1.0$$

$$-1.570796328$$

Проведем аналогичное вычисление для паде-аппроксимации синуса; при этом используем команду **unapply** для превращения выражения в функцию, так как команда **infnorm** оценивает норму функции.

```
> infnorm(unapply(p,x),-2..2, 'xmax'); xmax;
```

$$.9959222950$$

$$-1.554010892$$

Далее обратимся к команде **minimax** для получения наилучшей аппроксимации синуса полиномом пятой степени.

```
> minimax(sin(x), x=-2..2,5);
```

$$-.102427\,10^{-8} + (.9987651408 + (-.2730800000\,10^{-9} + (-.1641538513 + (.2284050000\,10^{-9} + .007042122100\,x)x)x)x)x)x$$

Наконец, используем для аппроксимации полиномы Чебышева, причем для сокращения числа цифр уменьшим мантиссу.

```
> Digits:=5: ch:=chebpade(sin(x),x,3);
> chebpade(sin(x),x,[3,3]);
```

$$ch := .88010\,T(1,x) - .039127\,T(3,x)$$

$$\frac{.89083\,T(1,x) - .027887\,T(3,x)}{T(0,x) + .025529\,T(2,x)}$$

7.12. Комбинаторика

Элементарные комбинаторные операции можно проводить при помощи команд стандартной библиотеки. Так, при помощи команды **binomial(n,r)** вычисляются биномиальные коэффициенты

$$C_n^r = \frac{n!}{r!(n-r)!}.$$

Если $0 \le r \le n$, то это число выборов r объектов из n объектов, для всех других случаев используется формула

$$C_n^r = \frac{\Gamma(n+1)}{\Gamma(r+1)\Gamma(n-r+1)} .$$

Если в качестве аргументов заданы символьные переменные, то за исключением простых случаев типа C_n^l возвращается неоцененное выражение. Приведем простые примеры.

```
> binomial(5,2);
```

$$10$$

```
> binomial(n,2);
```

$$\text{binomial}(n, 2)$$

```
> expand(");
```

$$\frac{1}{2}(n-1)\,n$$

Множество разных команд собрано в комбинаторный пакет **combinat**. К его командам можно обращаться по имени

command(args),

если предварительно загружен весь пакет, или, указывая префикс пакета перед командой,

combinat[command](args).

Перечислим некоторые команды пакета. Для вычисления сочетаний и числа сочетаний применяются команды **choose** и **numbcomb**, а для вычисления перестановок и их числа имеются команды **permute** и **numbperm** соответственно; команда **fibonacci** служит для определения **n**-го числа Фибоначчи.

Приведем пример использования перечисленных команд:

```
> with(combinat):
> choose([a,b,c]); numbcomb([a,b,c]);
```

$$[[\,], [a], [b], [a, b], [c], [a, c], [b, c], [a, b, c]]$$

$$8$$

```
> permute([a,b,c]); numbperm([a,b,c]);
```

$$[[a, b, c], [a, c, b], [b, a, c], [b, c, a], [c, a, b], [c, b, a]]$$

6

7.13. Группы и формы

Пакет **difforms** предоставляет команды для создания дифференциальных форм и работы с ними. Например, можно определить базис при помощи команды **defform** и использовать операции внешнего умножения (**&^**) и производной (**d**).

Для получения сведений об используемых пакетом типах следует обратиться к справке **?difforms[type]**, где в качестве **type** могут выступать **const**, **scalar** или **form**.

В пакете **liesymm**, позволяющем работать с симметриями Ли, используются своя внешняя производная (**d**) и внешнее произведение (**&^**), не зависимые от имеющихся в пакете **difforms**. Определение списка координатных переменных (нуль-форм) производится при помощи команды **setup**, для вычисления производной Ли служит команда **Lie**, команда **wcollect** применяется для представления формы в виде суммы форм. Для преобразования форм и представления результатов имеются команды **choose**, **getcoeff**, **mixpar**, **wdegree**, **wedgeset** и **value**.

Команды пакета не требуют непосредственной работы с дифференциальными формами. Если задан набор дифференциальных уравнений, то при помощи команды **determine** определяются и координаты и дифференциальные формы. Для обозначения частных производных лучше использовать команду **Diff**, чем команды **diff** и **D**. Для определения нужного порядка смешанных производных можно использовать команду **mixpar**, а для конвертирования производных **Diff** в производные **diff** при представлении результатов работы команды **determine** нужно использовать команду **value**.

В пакете **group** собраны команды, реализующие операции работы с группами.

7.14. Другие пакеты

Команды пакета **totorder** позволяют упорядочивать переменные согласно заданным неравенствам. Команда **tassume** вводит отношения между именами, команда **ordering** выдает текущее упорядочение. Команда **tis** служит для определения истинности высказывания о неравенстве. Команда **forget** уничтожает отношения для

одной переменной, а **init** сбрасывает все назначения. Работу этого пакета вполне характеризует приводимый пример

```
> with(totorder): tassume( a<b,c<b, b=d);
```

$$assumed, a < b, c < b, b = d$$

```
> tassume(g>b,f=c); ordering(); tis(f<g);
```

$$assumed, b < g, f = c$$

$$a < f, f = c, c < b, b = d, d < g$$

true

Пакет **Gauss** является инструментальным средством для реализации сложных алгоритмов. Разработчики Maple считают, что заложенные в этом пакете возможности по конструированию "области вычислений" позволяют писать лучшие коды. Предполагается, что в будущих версиях Maple произойдут изменения, основанные на кодах этого пакета.

Вслед за фирмой Waterloo Maple Inc. и непосредственно М. Монаганом мы полагаем, что заинтригованный читатель посмотрит файл справки с соответствующим примером

```
?Gauss,example.
```

Кроме того, имеются следующие пакеты: **genfunc** – для работы с рациональными производящими функциями, пакет **GF** для определения конечного поля Галуа и операций над его константами и функциями, **grobner** для вычислений с использованием базиса Гребнера, пакет **NPspinor** для обслуживания формализма Ньюмана-Пенроуза.

Наконец, большое число всевозможных команд можно найти в свободно распространяемом программном обеспечении (shareware), которое либо включается в поставку фирмой Maple, либо написано многочисленными пользователями и распространяется по Internet. В гл. 11 "Заключение" приведена информация, которая поможет на начальном этапе освоения сетевого пространства Maple.

8. Программирование

О типах переменных, зарезервированных словах, операциях и стандартных функциях Maple-языка говорилось в предыдущих главах. Здесь речь пойдет о написании пользователем на Maple-языке собственных программ, процедур, создании библиотек. Для этого в пакете существует довольно широкий набор команд и конструкций, аналогичный существующим в известных языках программирования (например, в языке PASCAL). Ниже мы последовательно перечислим эти конструкции и команды, сопровождая изложение короткими примерами. Более содержательные примеры будут даны в гл. 10 "Мини-исследования".

8.1. Условный оператор

Условный оператор в Maple начинается с зарезервированного слова **if** и обязательно должен заканчиваться словом **fi** :

```
if bool then expr1 else expr2 fi;
```

Эта конструкция дает возможность в зависимости от значения логического условия **bool** выполнять выражение **expr1** (когда **bool1**=true) или выражение **expr2** (**bool1**=false). В качестве выражений **expr1** и **expr2** здесь может выступать любая последовательность Maple-команд. Отметим, что условный оператор может иметь сокращенный вид: **if bool then expr fi**; приведем пример:

```
> x:=2;
                                x:=2
> if x>2 then print(`x>2`); else x:=x^2;
>              print(2*x);fi:
                                 8
```

Для реализации сложных условий можно использовать полный вариант условного оператора, который имеет следующий вид:

```
if bool_1 then expr_1
elif bool_2 then expr_2 ...
elif bool_n then expr_n else expr_0 fi;
```

Таким образом, вложенность условий может быть практически неограниченной и реализуется при помощи вставки **elif**. В качестве выражений **expr_0**, **expr_1**,..., **expr_n** могут выступать любые последовательности Maple-команд. Приведем пример полного условного оператора:

```
> x:=7:
> if x<0 then x:=a; elif x=0 then x:=b;
>                   elif x<10 then x:=c;
>                   else x:=d; fi;
                        x := c
```

8.2. Операторы цикла

Перейдем к изложению существующих в Maple операторов цикла. Всего их четыре, а для их записи используются служебные слова **for**, **from**, **by**, **to**, **while**, **do** и **od**. Телом всех операторов цикла является последовательность команд, заключенных между **do** и **od**. Начнем с цикла типа перечисления, который есть практически во всех алгоритмических языках.

```
for var from val1 by val2 to val3 do expr od;
```

Тело цикла **expr** выполняется при каждом значении параметра цикла **var** , который изменяется от **val1** с шагом **val2** до тех пор, пока не станет больше **val3**. Отметим, что если шаг изменения **val2** равен единице, то оператор цикла допускает сокращенную форму: **for** var **from** val1 **to** val3 **do** expr **od;** приведем пример оператора цикла:

```
> for i from 0 by 4 to 8 do i; od;
                        0
                        4
                        8
```

Оператор цикла типа 'пока' в Maple имеет вид:

```
while bool do expr od;
```

Тело цикла expr выполняется, пока значение логического выражения bool истинно (**true**), и выполнение прекращается, если bool ложно (**false**). Приведем соответствующий пример:

```
> j:=0:
> while j<5 do j:=(j+1)^j od;
                                1
                                2
                                9
```

Следующий оператор цикла является некоторым симбиозом двух предыдущих:

```
for var from val1 by val2 while bool
    do expr od;
```

Тело цикла **expr** выполняется, пока логическое выражение **bool** является истинным, а переменная **var** изменяется от значения **val1** с шагом **val2**. Например:

```
> for x from 1 by 2 while x<6 do print(x) od;
                                1
                                3
                                5
```

Четвертый оператор цикла ориентирован на работу с аналитическими выражениями и имеет следующую форму:

```
for var in expr1 do expr2 od;
```

Тело цикла **expr2** (последовательность команд между **do** и **od**), выполняется, когда символьная переменная **var** последовательно принимает значения каждого из операндов алгебраического выражения **expr1**. Естественно, работа этой конструкции зависит от внутреннего представления выражения **expr1**. Если **expr1** сумма, то переменная **var** принимает поочередно значения каждого слагаемого, если произведение – то каждого сомножителя, и т.д. Поясним сказанное примером:

```
> f:=x^2+3*x+1/x; g:=simplify(f);
```

$$f := x^2 + 3x + \frac{1}{x}$$

$$g := \frac{x^3 + 3x^2 + 1}{x}$$

```
> for s in f do s; od;
```

$$x^2$$

$$3x$$

$$\frac{1}{x}$$

```
> for s in g do s; od;
```

$$x^3 + 3x^2 + 1$$

$$\frac{1}{x}$$

8.3. Процедуры-функции

Процедуры-функции в Maple можно задавать несколькими способами. Первые два аналогичны по смыслу и отличаются только синтаксисом. Один использует символ (->):

```
name:= (var1,var2,...) -> expr;
```

Здесь **name** – имя функции, **var1**, **var2**,... – имена формальных параметров, а **expr** – выражение, реализующее тело процедуры-функции. Нелишне напомнить, что типы формальных параметров и результата работы процедуры могут быть любыми. Другой вариант имеет вид:

```
name:=< expr | var1,var2,...>;
```

Приведем два примера задания процедур-функций.

```
> f:=(x,y)->simplify(x^2+y^2);
```

$$f := (x, y) \rightarrow \text{simplify}\left(x^2 + y^2\right)$$

```
> f(sin(x),cos(x));
```

$$1$$

```
> f:=<x+y|x,y>;
```

$$f := \langle x+y \mid x,y \rangle$$

```
> f(-x,x);
```

$$0$$

Второй способ задания процедуры-функции использует команду **unapply**

```
name:=unapply(expr,var1,var2,...);
```

Здесь **var1**, **var2**,... – переменные, а **expr** – выражение или операция. Эта команда полезна при определении новой функции через известную, или когда вычисленное выражение предполагается использовать как функцию. Например,

```
> f:=unapply(diff(z(x)^2,x)-2,z);
```

$$f := z \rightarrow 2\,z(x)\left(\frac{\partial}{\partial x} z(x)\right) - 2$$

```
> f(sin);
```

$$2\sin(x)\cos(x) - 2$$

8.4. Процедуры

Всякая процедура в Maple начинается с заголовка. Заголовок состоит из имени процедуры, за которым следует знак присваивания и служебное слово **proc**, затем в круглых скобках через запятую указываются формальные параметры. Процедура должна обязательно заканчиваться оператором **end**. Все команды и выражения, стоящие после заголовка и до оператора **end**, составляют тело процедуры. Простейшая процедура имеет следующий вид:

```
name:=proc( var1, var2, ...);
    expr1; expr2; ... end;
```

здесь **name** – имя функции, **var1**, **var2**, ... – имена формальных параметров, а **expr1**, **exp2**, ... – выражения, реализующие тело процедуры.

После того как процедура загружена, ее вызов осуществляется по имени. Возвращаемым значением по умолчанию является значение последнего оператора из тела процедуры. Тип результата работы процедуры зависит от типа возвращаемого значения. На-

пример, процедура с именем f, вычисляющая сумму переменных x и y, выглядит следующим образом:

```
> f:=proc(x,y);
> x+y;
> end:
```

$f := proc(x,y)\ x+y\ end$

Приведем результат обращения к процедуре.

```
> f(u,sin(v));
```

$$u + \sin(v)$$

Для написания процедур в Maple имеется ряд команд и служебных слов, кроме указанного выше обязательного минимального набора. Эти команды позволяют описывать переменные, управлять выходом из процедуры, сообщать об ошибках.

При описании формального параметра процедуры можно явно указывать его тип после двоеточия, следующего за именем параметра. При таком описании Maple автоматически проверяет тип фактического параметра и выдает сообщение об ошибке в случае его несовпадения с типом формального параметра.

После заголовка процедуры может следовать описательная часть процедуры, отделяющаяся от него пробелом. Для определения локальных, используемых только внутри данной процедуры, переменных применяется описатель **local var1, var2, ...**; здесь **var1, var2** – имена локальных переменных. Во избежание накладок с использованием имен рекомендуется описывать все переменные, встречающиеся в процедуре, в том числе и глобальные. Перечислить используемые глобальные переменные можно при помощи описателя **global**, который должен размещаться в описательной части процедуры.

После описателей в процедуре может стоять указатель на опции **options** opt_sec; выражение opt_sec принимает одно из следующих значений: **remember** (делает более эффективным выполнение рекурсивных процедур), **operator** (аналогичен заданию оператора-функции) и некоторые другие (**builtin**, **system**, **arrow**, **angle**, **trace**, **package** и **Copyright**).

Для выхода из процедуры в любом месте ее тела и присвоения результату ее работы различных величин в зависимости от условий

используется команда **RETURN(val)**. Здесь **val** – возвращаемое значение, которое может иметь различный тип при выходе из разных мест процедуры.

Для аварийного выхода из процедуры в случае возникновения ошибки и сообщения о случившемся используется команда **ERROR(`string`)**, где **string** – сообщение, которое должно появиться на экране в аварийной ситуации. Итак, схематично общий вид процедуры можно изобразить следующим образом:

```
name:=proc(par1,...,park) local val1,...,valn;
    global var1,...,varm; options opt_s;
        expr1; expr2;... RETURN(res1); ...
ERROR(`Error in procedure name`);...exprp; end;
```

Приведем пример, использующий все возможности для конструирования процедур. Описываемая процедура преобразует отрицательное число в положительное. Если входным параметром является нуль, то выдается сообщение об ошибке; для положительных чисел выдается массив, содержащий само число, его квадрат и куб.

```
> examp:=proc(x:float) global z; local y,w;
> options remember;
> if x<0 then RETURN(-x);
> elif x=0 then ERROR(`Variable x=0`); fi;
> [x,x^2,x^3];
> end;
```

```
    examp :=
      proc(x:float)
      local y,w;
      global z;
      options remember;
        if x < 0 then RETURN(-x)
        elif x = 0 then ERROR(`Variable x=0`)
        fi;
        [x,x^2,x^3]
      end
```

```
> examp(-1);
```

```
                                  1
```

```
> examp(0);
```

```
                    Error, (in examp) Variable x=0
> examp(3);
                                [3, 9, 27]
```

8.5. Команды ввода/вывода

В гл. 1 "Среда Maple" речь шла о сохранении файла всего сеанса работы на диск, но часто бывает нужно сохранить только конечный результат работы, а не весь ход решения. Это становится особенно актуальным, если для получения результата проводился большой объем символьных вычислений и манипуляций с формулами. Для этих целей в Maple существует команда **save**. Если результатами работы являются значения переменных **var1**, **var2**,..., то для сохранения их в файл с именем **name** и расширением **ext** нужно дать команду:

```
save var1,var2,...,`name.ext`;
```

Представление сохраняемой информации в файле зависит от расширения **ext**. Если в качестве расширения указано **'m'**, то файл запишется во внутреннем Maple-формате, при всех других расширениях в текстовом формате запишутся команды Maple.

Для ввода сохраненной информации из файла с именем **name** и расширением **ext** используется команда

```
read `name.ext`;
```

Для записи результатов работы в файл имеются две команды:

```
writeto(`name.ext`)
```

и

```
appendto(`name.ext`).
```

После этих команд все вводимое и результаты будут записаны в файл с именем **name**. Тип файла по-прежнему зависит от указываемого расширения **ext**. Разница между двумя указанными командами в том, что если вызвана первая команда, то информация записывается с начала файла, а в случае вызова второй – информация дописывается в конец файла. Для восстановления выдачи на экран нужно повторно обратиться к команде **writeto**, указав в качестве имени файла **terminal** (т.е. после команды **writeto(`terminal`)** все результаты снова выводятся на экран).

Существует также возможность записи результатов в файл при помощи следующих команд:

open(`name.ext`) – открыть файл с именем name и расширением ext.

write(expr1,expr2,...) – записать выражения **expr1**, **expr2**,... в открытый для записи файл.

writeln(expr1,expr2,...) – записать выражения **expr1**, **expr2**,..., начиная с новой строки в открытый для записи файл.

close(`name.ext`) – закрыть файл **name.ext**.

Часто при написании программы необходимо выводить информацию о ходе решения, результаты, аварийные сообщения и пр. Для этих целей в пакете предусмотрен ряд команд печати. Наиболее простой является команда **print**, обращение к которой имеет вид:

print(expr1,expr2,...,exprn),

здесь **expr1**,**expr2**,...,**exprn** – любые Maple-выражения. Если переменной ничего не присвоено, то печатается просто имя переменной, в противном случае печатается ее содержимое. Приведем пример обращения к команде печати:

```
> x:=y^2: print(x,`Information`,y,factor(x-3*y)):
```

$$y^2,\ Information,\ y,\ y\,(y-3)$$

В отличие от команды **print**, которая печатает выражения через запятую в естественном математическом виде, команда **lprint(expr1,expr2,...)** выводит информацию в стиле строки ввода и разные выражения отделяются друг от друга тремя пробелами. Например:

```
> x:=y^2: lprint(x,`Information`,y,factor(x-3*y)):
```

```
y^2   Information   y   y*(y-3)
```

Кроме команд бесформатного вывода **print** и **lprint** в Maple есть еще команда **printf(fmt,expr1,expr2,...)**; здесь **fmt** – спецификация формата вывода, полностью идентичная аналогичной команде языка C.

Для удобства использования в других программах результатов аналитических преобразований в пакете Maple предусмотрена возможность представлять и сохранять выражения в следующих форматах: в формате языка C, в формате языка FORTRAN, в формате LaTeX. Вывод выражения в виде FORTRAN-операторов осуществля-

ется по команде **fortran(expr,options)**, где в качестве **expr** могут выступать выражение, массив выражений или список уравнений. Параметры **options** могут принимать следующие значения: **filename=`name.ext`** (результат выводится в файл), **optimized** (FORTRAN-выражения оптимизируются, вводятся вспомогательные переменные), **digits** и **mode** (задание точности вычислений).

Чтобы записать выражение на языке C необходимо подключить нужную команду при помощи **readlib**(C), после чего можно пользоваться командой **C(expr,options)**, с параметрами, аналогичными описанным для команды **fortran**.

Для перевода выражения expr в формат системы редактирования LaTeX используется команда **latex(expr)** или **latex(expr,`name.ext`)**. Во втором варианте результат выводится в файл.

Теперь приведем пример использования некоторых из перечисленных команд:

```
> x:=a+b^5*sin(b^3);
```

$$x := a + b^5 \sin\left(b^3\right)$$

```
> fortran(x,optimized,filename=`res.for`)
> readlib(C): C(x);
```

```
t0 = a+pow(b,5.0)*sin(b*b*b);
```

```
> readlib(write): open(`res.ltx`);
> latex(`Expression x:`); writeln(); latex(x);
> close(`res.ltx`);
```

В результате этих команд будет создан файл 'res.for' со следующим текстом:

```
t1 = b**2
t2 = t1**2
t7 = a+t2*b*sin(t1*b)
```

и файл 'res.ltx' с текстом в формате LaTeX:

```
\mbox {{\tt `Expression x:`}}
a+{b}^{5}\sin({b}^{3})
```

Пакет Maple можно использовать для анализа и графической интерпретации числовой информации, находящейся в текстовом файле и полученной как при помощи самого пакета, так и других

программ. Обычно в текстовом файле числа записаны по строкам, по нескольку чисел в строке. Для считывания числовой информации из файла используется команда:

```
readdata(name,options,posint),
```

где **name** – имя файла, **options** – тип переменных (**integer**/**float** – установка по умолчанию), **posint** – счетчик чисел (сколько считывать чисел из строки). Считанная информация представляется в виде переменной типа **list**. Перед использованием команду **readdata** необходимо подключить при помощи **readlib(readdata)**.

Например, пусть в текстовом файле *'a.txt'* в строке находятся числа: *1 2 3 4 5 6 7 8 9*. Считаем данную информацию из файла и занесем ее в переменную *data*.

```
> readlib(readdata):
> data:=readdata(`a.txt`,integer,9);
                    data:=[[1,2,3,4,5,6,7,8,9]]
```

Двойная индексация у переменной *data* связана с тем, что числа записываются в двумерный массив, так что число строк массива равно числу считанных строк, а число столбцов определено параметром **posint**.

Для считывания строки из файла *name.ext* применяется команда **readline(`name.ext`)**. Результат действия команды присваивается переменной типа **string**. Если в качестве имени файла указать **terminal**, то программа ожидает ввода с экрана.

Кроме того, в пакете **stats** имеется команда **readstat** для ввода статистической информации.

8.6. Создание собственных библиотек

Здесь речь пойдет о создании собственного пакета. В создаваемый пакет имеет смысл помещать уже отлаженные и оттестированные процедуры. Напомним, что один из вариантов обращения к командам, находящимся в пакетах, выглядит следующим образом: **name_pack[name_com](options)**, где **name_pack** – имя пакета, **name_com** – имя команды. Полное имя команды, входящей в пакет, состоит из имени этого пакета и имени самой команды. Таким обра-

зом пакет представляет собой как бы массив, элементами которого являются команды.

Для создания пакета нужно процедуру присвоить переменной с составным именем. После того как пакет написан, его нужно сохранить на диске в файле с расширением 'm' (например, с именем **namefile**) при помощи команды **save name_pack,`namefile.m`**. В последующих сеансах, перед подключением, пакет предварительно нужно считывать с диска при помощи команды **read `namefile.m`**. Для подключения пакета используется команда **with**.

В Maple имеется возможность сопровождать написанные команды текстами справки. Для этого используются команды **TEXT** или **makehelp**. Для создания справки к команде **my_com** при помощи **TEXT** используется конструкция: **`help/text/my_com`:= TEXT(str1,str2,...)**, где **str1**, **str2**,... – строки текста справки.

Формат команды **makehelp** иной: **makehelp(topic, txtfile, library)**, здесь **topic** – имя команды, для которой создается справка, **txtfile** – имя текстового файла, содержащего текст справки, **library** – имя библиотеки, содержащей команду.

Приведем пример написания небольшого пакета. Создадим пакет *exam_pack* из трех процедур с именами *proc1, proc2, proc3* и снабдим его справкой. Пакет сохраняется в файле *'exam_pack.m'* и подключается, после чего становятся доступны все его команды.

```
> exam_pack[proc1]:=
> proc(a) local s; s:=2*a; print(`proc1: `,s);
> RETURN(s); end:
> exam_pack[proc2]:=
> proc(a) local s; s:=a^2; print(`proc1: `,s);
> RETURN(s); end:
> exam_pack[proc3]:=
> proc(a,b) local s; s:=(a+b)/2;
>  print(`proc3: `,s);
>  RETURN(s); end:
> `help/text/exam_pack` :=TEXT(
> `Это пример создания собственной библиотеки`,
> `В библиотеку входит три процедуры:`,
> `proc1 - умножает входной параметр на 2,
>  печатает и возвращает`,
> `proc2 - возводит входной параметр
```

```
> в квадрат, печатает и возвращает`,
> `proc3 - считает полусумму величин a и b`);
> save exam_pack,`exam_pack.m`;
```

Если нужно использовать процедуры пакета в другой программе, то пакет нужно считать с диска командой **read**.

```
> read `exam_pack.m`:
> with(exam_pack):
```

Если пользователем написаны и используются несколько собственных пакетов, то целесообразно создать собственную библиотеку. В комплекс программ Maple входит ряд выполняемых программ, одна из которых **march.exe** является сервисной программой для создания библиотек. С помощью этой программы пользователь может создавать библиотеки, добавлять и удалять пакеты.

8.7. Отладка программ

В этом пункте мы рассмотрим возможности получения более подробной информации о стандартных Maple-командах и ходе вычислений, используемых ресурсах компьютера и ошибках, перечислим команды управления выводом результатов.

Начнем с системной переменной **printlevel**, которая задает детальность печати сообщений о ходе преобразований. Если значение этой переменной находится в интервале от 1 до 5, то выводится информация о данной команде, если от 6 до 10 – то и о вызываемой вложенной команде первого уровня, и т.д. Для того чтобы выводилась информация о работе команды с уровнем вложенности k, значение **printlevel** должно быть равно 5*k. По умолчанию **printlevel**=1, а это значит, что выводится результат только непосредственно вызываемой команды. При значении **printlevel**=100 выводится информация о работе всех команд.

Для получения подробной информации о работе некоторой команды **comm** можно использовать вызов **trace(comm)**; после этой команды при обращении к **comm** будет выводиться информация о ее работе. Отменить действие команды **trace** можно при помощи команды **untrace** с аналогичными параметрами. Например:

```
> trace(diff);
> printlevel:=5;
```

```
> diff(x*sin(x),x);
```

{--> enter sin, args = x

<-- exit sin (now at top level) = sin(x)}

*{--> enter diff, args = x*sin(x), x*

{--> enter diff, args = x, x

$$1$$

<-- exit diff (now in diff/sin) = 1}

$$\sin(x) + x\cos(x)$$

*<-- exit diff (now at top level) = sin(x)+x*cos(x)}*

$$\sin(x) + x\cos(x)$$

Управлять представлением результата работы команд на экране можно при помощи команды **interface(arg1,arg2,...)**, где параметры **arg** имеют вид **name=val**. Приведем только некоторые возможные значения переменных **name** и **val**. Беря в качестве **name** имя **prettyprint** и задавая число **val**, пользователь может управлять видом стандартных сообщений на экране (**prettyprint=0** – используется команда **lprint**). Указание **verboseproc=n** задает степень подробности печати текстов процедур: при n=0 текст не печатается, при n=1 печатается только текст пользовательских процедур, при n=2 печатается текст всех процедур. Если в качестве **name** используется **plotdevice**, то в зависимости от значения строковой переменной **val** можно указать тип вывода графики (**gif** – в формате gif, **laserjet** – в формате HP LaserJet принтера и т.д).

Указать имя файла для вывода графики можно при помощи конструкции: **plotoutput=`namefile.ext`**. Например, после команд:

```
> interface(plotdevice=gif,plotoutput=`ggg.gif`);
> plot3d(sin(x*y)*y,x=-4..4,y=-4..4);
```

на диске появится файл *ggg.gif* с графической копией экрана в формате GIF. Обратим внимание на то, что, пока значение **plotoutput** не будет переопределено, результаты всех команд графики будут записываться в один файл.

Часто бывает интересным, а иногда и необходимым знать тексты стандартных Maple-команд. Вывести на экран текст нужной процедуры, реализующей команду **comm**, можно при помощи команды **print(comm)**. Перед этим нужно определить параметры вывода командой **interface**. Приведем набор команд, в результате которых выводится текст команды **collect**.

```
> interface(verboseproc=2,prettyprint=1,version);
                              SCG2
> print(collect);
    proc(p,x)
    local t,form,fun,v;
    options `Copyright 1991 by the University of Waterloo`;
      if type(p,{relation,set,list}) then RETURN(map(collect,args))
    fi;
      if not type(p,algebraic) then ERROR(`invalid 1st argument`,p)
    fi;
      if type(x,{set,list}) then v := op(x) else v := x fi;
      for t in [v] do if not type(t,{function,name})
       then ERROR(`cannot collect`,t) fi od;
      form := 'recursive';
      fun := <x|x>;
      for t in [args[3 .. nargs]] do
        if t = 'recursive' then next
        elif t = 'distributed' then form := 'distributed'
        else fun := t
        fi
      od;
      if form = 'recursive' then `collect/recursive`(p,op(fun),v)
      else `collect/distributed`(p,op(fun),1,v)
      fi
    end
```

Для обработки ошибок вычислений в Maple предусмотрены команда **traperror** и системная переменная **lasterror**. В переменной **lasterror** сохраняется сообщение о последней случившейся ошибке, причем содержимое переменной очищается при обращении к **traperror**. Поясним сказанное примером:

```
> 1/0:
    Error, division by zero
> lasterror; 1/4; lasterror;
                          division by zero
                                1/4
                          division by zero
> traperror(1/4);lasterror;
```

$$\frac{1}{4}$$

lasterror

Существует ряд команд, позволяющих следить за использованием ресурсов компьютера. Так, команда **time**() показывает время процессора, используемое с начала текущего сеанса работы с Maple. Время выполнения каждой команды будет показываться после запуска команды **showtime**(), которую нужно предварительно подгрузить командой **readlib**. Количество используемой памяти можно посмотреть при помощи команды **words**().

Конечно, нами перечислены далеко не все возможности по отладке и диагностике программ. Так, в поставку пакета Maple входит программа **mint.exe**, которая позволяет диагностировать написанные программы, анализировать синтаксические ошибки, использование переменных и оптимальность написанных процедур. При обращении к ней нужно указать соответствующие ключи (см. справку **mint**) и имя файла, содержащего исходный текст Maple-программы.

9. Графика в Maple

Одним из интересных и эффектных применений пакетов компьютерной алгебры является использование их графических возможностей при решении различных задач: для визуализации результатов исследований, графической интерпретации данных и т.д. Если даже отбросить все другие возможности Maple и изучить только графические команды, то эти знания позволят пользователю эффективно применять возможности компьютерной графики от рисования простого графика функции до создания мультфильмов.

Исходными данными для построения графических образов могут быть конструкции Maple (функции, массивы, графические структуры), а также результаты вычислений, полученные при помощи других программ и записанные в текстовом файле. В качестве параметров при вызове графических команд указывается выводимый объект (функция, набор точек, множество), интервалы изменения переменных и опции вывода, управляющие видом изображения.

В пакете Maple имеется богатый набор команд двумерной и трехмерной графики. В этой главе в основном речь пойдет о командах из библиотеки **plots**. Ряд графических команд, ориентированных на решение специальных задач, находится в других библиотеках, и о некоторых из них будет сказано в конце этой главы и в других главах.

Наиболее универсальные и часто используемые команды графической библиотеки **plots** доступны пользователю по умолчанию. Перед обращением к другим командам нужно или подключить всю библиотеку

```
with(plots),
```

или использовать при вызове конкретной команды конструкцию

```
plots[имя команды](параметры).
```

Результатом правильно введенной команды будет построение рисунка в отдельном Windows-окне, имеющем свое меню (меню графики). При помощи пунктов этого меню можно интерактивно изменять построенное изображение (менять ракурс, цвет и др.). Однако из этого меню возможно только частичное управление видом изображе-

ния, а полное управление реализуется при назначении опций графической команды. В Windows-версии для переноса полученного рисунка в текст Maple-документа нужно выполнить следующие действия: занести образ в буфер, нажав комбинацию клавиш CTRL-C или выбрав этот пункт в меню графики, затем перейти в режим работы с документом и скопировать содержимое буфера, нажав CTRL-V или указав пункт PASTE основного меню. Подробнее о пунктах меню можно прочитать в Приложении.

По умолчанию устройством вывода графики является монитор, но часто рисунки нужно скопировать, вставить в статьи и отчеты, сохранить на диске в понятном другим программам формате. Все перечисленные пожелания можно исполнить при помощи команды **interface** и двух ее опций **plotdevice** и **plotoutput** (см. гл. 8 "Программирование").

Обратим внимание на то, что при обращении к графическим командам в выражениях, которые задают рисуемые объекты, не должно быть неопределенных переменных. Если это не так, то Maple выдаст сообщение о невозможности построения графика. Для вывода графиков нескольких однотипных выражений на одном рисунке достаточно эти выражения заключить в фигурные скобки и указать в качестве параметра графической команды.

Ниже подробно рассмотрены основные команды, опции и структуры двумерной и трехмерной графики, но, перед тем как перейти к их изложению, приведем без комментариев самый простой вариант рисования графика функции одной переменной f(x) в интервале a<x<b:

```
plot(f(x),x=a..b);
```

9.1. Опции двумерной графики

Почти все опции двумерной графики, за исключением специализированных, применимы для всех рассматриваемых команд и позволяют управлять представлением изображения: детальностью графика, типом выводимых линий и заполнителей, размещением надписей и т.д. Опции следуют сразу за обязательными параметрами в командах, причем в случае их отсутствия используются установки по умолчанию (в тексте отмечены подчеркиванием). Отметим, что пользователь может переопределять установки на сеанс при помощи ко-

манды **setoptions(options)**. Перечислим и прокомментируем основные опции:

title=`Name` – заголовок рисунка.

coords=polar – тип координат (полярные или декартовы).

axes=val – тип выводимых осей координат. Величина **val** может принимать одно из значений: **NORMAL** – обычные оси координат, **BOXED** – график заключается в рамку с нанесенной шкалой, **FRAME** – оси с центром в левом нижнем углу рисунка, **NONE** – вывод без нанесения осей.

scaling=val – тип масштаба рисунка. Если **val** принимает значение **CONSTRAINED**, то график выводится с одинаковым масштабом по осям координат, а при значении **UNCONSTRAINED** – график масштабируется по размеру графического окна.

style=LINE/POINT – вывод линиями или точками.

numpoints=n – число вычисляемых точек графика (по умолчанию равно 49).

resolution=n – горизонтальное разрешение дисплея в пикселях (по умолчанию 200).

color=colorvalue – цвет вывода. В качестве **colorvalue** может выступать одно из следующих зарезервированных в Maple названий цветов:

aquamarine, black, blue, navy, coral, cyan, brown, gold, green, gray, grey, khaki, magenta, maroon, orange, pink, plum, red, sienna, tan, turquoise, violet, wheat, white, yellow,

кроме того, существует возможность определения собственных цветов.

xtickmarks=nx – число насечек по оси абсцисс (X).

ytickmarks=ny – число насечек по оси ординат (Y).

thickness=n – толщина линии; здесь **n** может принимать значения 1,2,3,....

linestyle=n – тип выводимой линии (непрерывная, пунктир, ...), по умолчанию – непрерывная линия при n=1.

symbol=s – тип символа, которым помечаются точки, здесь s может принимать одно из следующих значений: **BOX**, **CROSS**, **CIRCLE**, **POINT**, **DIAMOND**.

font=[vfon,style,size] – задание шрифта для вывода текста. Здесь переменная **vfon** задает имя шрифта (**TIMES**, **COURIER**, **HELVETICA** или **SYMBOL**), переменная **style** определяет стиль шрифта (см. справку Maple), а число **size** – размер символа.

labels=[str_X,str_Y] – надписи по осям координат.

Выше перечислены важнейшие универсальные, общие для всех команд, опции двумерной графики, об остальных можно узнать, обратившись к справке. Укажем еще несколько специфических, но важных опций. Для команд **fieldplot** и **gradplot** существует дополнительная опция, которая задает тип стрелок для изображения вектора:

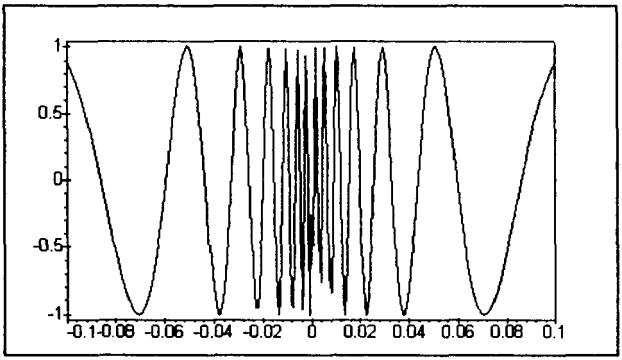
arrows= **thin** (или **line**,**slim**,**thick**).

Для команд **fieldplot**, **gradplot** и **implicitplot** существует опция, задающая размеры сетки для вычисления векторного поля и неявно заданной функции на плоскости:

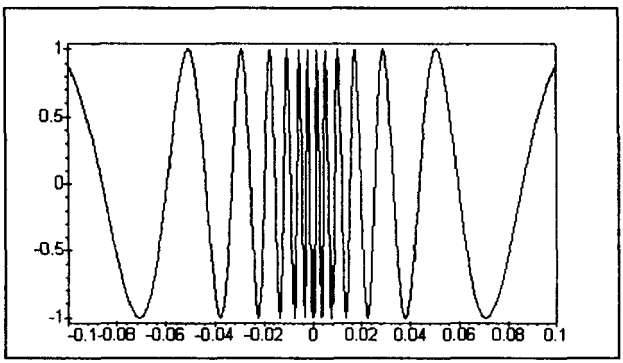
grid=[int1,int2], где **int1** – число узлов по оси X, **int2** – по оси Y. По умолчанию принято **grid=[20..20]**.

В Windows-версии многие опции вывода могут переопределяться интерактивно в меню графики (см. Приложение). Отметим, что в меню изменяются только внешние атрибуты рисунка (тип осей, вид масштаба и пр.), а наиболее важные опции можно определить только при задании команды. В графическом меню нельзя вставить или отредактировать надписи, изменить число узлов, поменять цвет линий и т.д. Более того, неправильное или невнимательное назначение опций может привести к неточностям. Приведем пример, иллюстрирующий, что может получиться, если при построении графика сильно осциллирующей функции использовать число узлов по умолчанию (первый рисунок) и как изменится картинка при увеличении их числа (второй рисунок).

```
> plot(sin(1/(abs(x)+0.02)),x=-0.1..0.1);
```




```
> plot(sin(1/(abs(x)+0.02)),
>      x=-0.1..0.1,numpoints=999);
```



Примеры использования описанных опций будут даны ниже вместе с примерами действия команд двумерной графики.

9.2. Команды двумерной графики

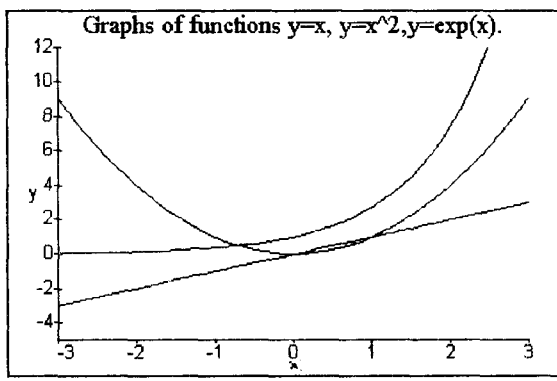
Основной командой двумерной графики является команда **plot**, которая находится в стандартной библиотеке и не требует предварительного вызова. В зависимости от представления входных параметров она позволяет рисовать графики функций одной переменной, параметрически заданных функций, набора точек и т.д.

Начнем с построения графиков функций одной переменной, заданных явной формулой. В этом случае команда **plot** имеет вид:

```
plot({func1,func2...},x=a..b,y=c..d,<options>);
```

здесь **func1,func2,...** – выражения, зависящие от переменной **x**, **a..b** – интервал изменения переменной **x** (отрезок оси абсцисс), **c..d** – выводимый интервал по оси ординат. Здесь и далее в угловых скобках указаны аргументы, которые могут отсутствовать. Если одним из концов интервала изменения **x** является бесконечность, то выводится график асимптотического поведения функции. Приведем пример построения графиков трех функций, используя опции для явного указания числа насечек по осям координат, вида осей координат, цвета линий и заголовка:

```
> plot({x,x^2,exp(x)},x=-3..3,y=-5..12,
>     title=`Graphs of functions y=x,y=x^2,y=exp(x)`,
>      axes=FRAME,xtickmarks=7,
>      ytickmarks=9,color=black);
```

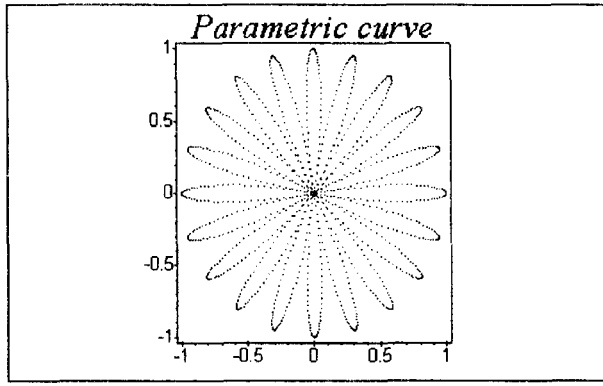


Для вывода параметрически заданной кривой используется следующий формат команды **plot**:

plot([funx(t),funy(t),t=a..b],<options>),

где **funx(t)**, **funy(t)** – функции координат, зависящие от параметра **t**; **a** и **b** – интервал изменения параметра. Например:

```
> plot( [cos(10*t),t,t=0..2*Pi], numpoints=1200,
>    title =`Parametric curve`, axes= boxed,
>    style=point, symbol=POINT, coords=polar,
>    scaling=CONSTRAINED,
>    titlefont=[TIMES,ITALIC,20]);
```



В приведенном примере использованы следующие опции: **numpoints** – для указания числа точек при построении кривой, **title** – для введения заголовка, **axes** – для определения типа осей координат, **style** и **symbol** – для задания вида изображаемой кривой, **coords** – для указания полярной системы координат, **scaling** – для изображения с одинаковым масштабом по осям, **titlefont** – для определения шрифта заголовка.

Для изображения набора точек используется команда:

plot([x1,y1,x2,y2,...],x=a..b,y=c..d,<options>),

где **[x1,y1,x2,y2,...]** – набор точек. Этот набор может также задаваться в виде: **[[x1,y1],[x2,y2],...,]**, здесь **x1,x2,...** – абсциссы, а **y1,y2,...** – ординаты.

Отметим, что команда **plot** имеет и другие возможности для построения графиков самых разнообразных Maple–выражений.

Кроме многофункциональной команды **plot** в пакете имеются следующие команды двумерной графики:

logplot(expr,var1=a..b,<options>) – построение графика выражения **expr** в логарифмическом масштабе по оси Y.

loglogplot(expr,var1=a..b,<options>) – построение графика выражения **expr** в логарифмическом масштабе как по оси X, так и по оси Y.

poligonplot([pt1,pt2,...],<options>) – построение n-угольника, заданного вершинами **pt1,pt2,...**, причем последняя точка соединяется с первой.

conformal(F,r1,r2,<options>) – изображение конформного отображения комплекснозначной функции **F**.

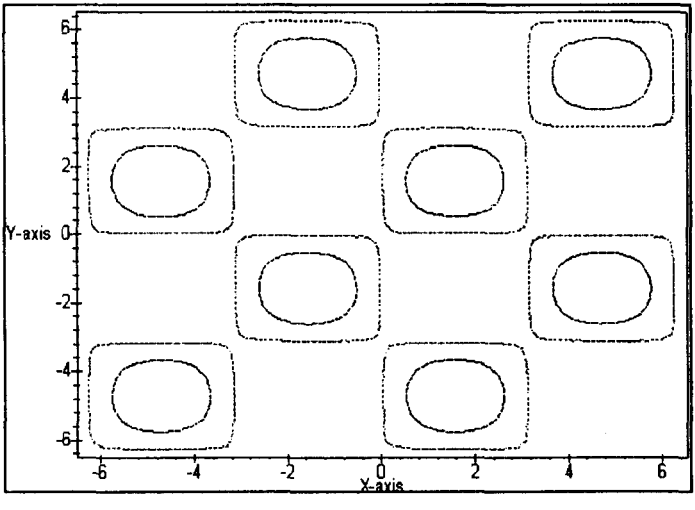
textplot([exprx,expry,string],<options>) – вывод текстовой строки **string**, начиная с точки с координатами **exprx**, **expry**.

polarplot([rad,ang,var=a..b],<options>) – вывод графика в полярных координатах, где функция радиуса **rad** и угла **ang** зависят от переменной **var**, изменяющейся на отрезке [**a**,**b**].

implicitplot(ex=f,var1=a..b,var2=c..d,<options>) – изображение линии уровня значения **f** для функции **ex** в прямоугольнике [**a**,**b**]×[**c**,**d**].

Приведем пример построения двух линий уровня, используя для большей детализации опцию **grid**. Опция **labels** позволяет снабдить оси координат надписями.

```
> implicitplot({sin(x)*sin(y)=1/40,
>              sin(x)*sin(y)=1/2}, x=-2*Pi..2*Pi,
>            y=-2*Pi..2*Pi,axes=BOXED,grid=[70,70],
>              labels=[`X-axis`,`Y-axis`]);
```



densityplot(expr,var1=a..b,var2=c..d,<options>) – рисование функции плотности линий уровня для выражения **expr**, зависящего от переменных **var1** и **var2**.

fieldplot([expr1,expr2],x=a..b,y=c..d,<options>) – построение векторного поля, определяемого выражениями **expr1** и **expr2**; переменные **x** и **y** изменяются на отрезках [**a**,**b**] и [**c**,**d**] соответственно.

sparsematrixplot(A,<options>) – изображение ненулевых элементов матрицы A. Команда полезна для визуализации матриц, возникающих в численных приложениях (методы конечных разностей и конечных элементов).

В следующем примере при помощи команд из пакета линейной алгебры **linalg** случайным образом формируется матрица A, которая затем приводится к треугольному виду. После этого двум элементам присваиваются ненулевые значения и полученная матрица B изображается командой **sparsematrixplot**.

```
> with(plots): with(linalg):
> A:=randmatrix(30,30): B:=gausselim(A);
> B[10,1]:=1/10: B[28,21]:=1:
> sparsematrixplot(B,symbol=DIAMOND);
```

gradplot(expr,var1=a..b,var2=c..d,<options>) – изображение векторного поля, задаваемого градиентом выражения **expr**, вычисляемого по переменным **var1**, **var2**.

odeplot(s,vars,r1,r2,<options>) – изображение решения дифференциального уравнения (см. пример в гл. 6 "Дифференциальные уравнения").

replot(p,<options>) – построение рисунка для графической структуры **p**, в которой изменен ряд опций.

Часто бывает нужно совместить на одном рисунке графические образы, полученные при помощи различных графических команд. Для этого результат действия каждой команды должен быть присвоен некоторой переменной. При этом вывода на экран не происходит. Для вывода полученных графических образов служит команда:

display([pic1,pic2,...],<options>)

Здесь образы **pic1**,**pic2**,... выводятся на одном рисунке в общих осях координат. При указании опции **insequence=true** образы **pic1**,**pic2**,... будут показываться один за другим, составляя кадры мультфильма.

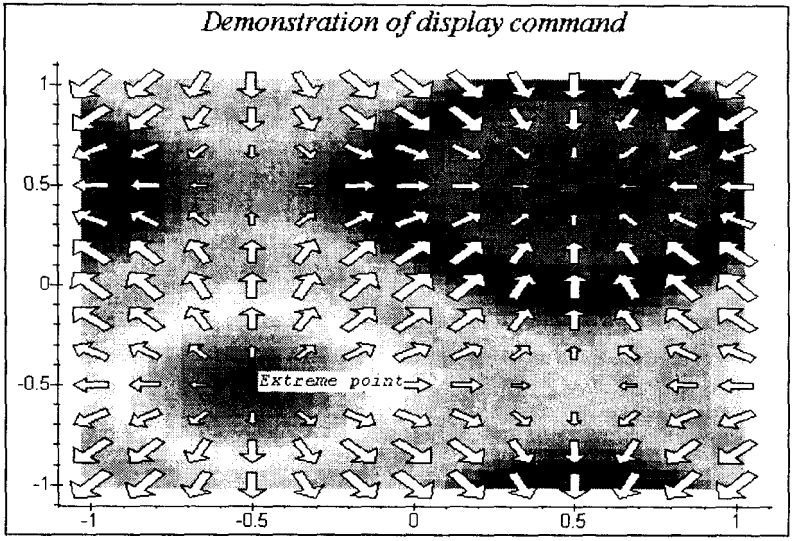
Для двумерной мультипликации может применяться также команда:

animate(F,x=a..b,t=c..d,frame=n) – здесь **F** – выражение, зависящее от переменных **x** и **t**, которые изменяются соответственно в интервалах [**a**,**b**] и [**c**,**d**]. Переменная **x** отвечает оси абсцисс, **t** – переменная времени, а опция **frame** задает число кадров.

Работу некоторых перечисленных команд продемонстрируем примером, в котором на одном рисунке совмещаются векторное поле градиента функции *f*, изображение плотности линий уровня этой функции без нанесения сетки (опция **style**), вывод предварительно вычисленной точки экстремума функции *f*, которая снабжается надписью.

```
> with(plots):
> f:=sin(Pi*x)+1.2*sin(Pi*y):
> pic1:=gradplot(f,x=-1..1,y=-1..1,grid=[13,13],
> arrows=THICK,axes=FRAMED):
> pic2:=densityplot(f,x=-1..1,y=-1..1,
```

```
> extr[1]:=rhs(uu[2]): extr[2]:=rhs(uu[1]):
> pic3:=plot([extr[1],extr[2]],style=POINT,
>         symbol=DIAMOND,color=black):
> pic4:=textplot([extr[1]+0.02,extr[2]+0.01,
>         `Extreme point`],color=blue,
>     font=[COURIER,OBLIQUE,10],align = {RIGHT}):
> display([pic2,pic1,pic4,pic3],axes=FRAMED,
>       titlefont=[TIMES,ITALIC,16],
>     title=`Demonstration of display command`);
```



9.3. Двумерные графические структуры

Команды, о которых речь шла выше, в основном преобразуют входные данные в графические структуры и выводят их на экран при помощи команды рисования PLOT.

Перечислим двумерные графические структуры:

`POINTS([x1,y1],[x2,y2],...)` – набор точек с координатами `[x1,y1]`, `[x2,y2]`...

POINTS([x1,y1],[x2,y2],...) – набор точек с координатами **[x1,y1]**, **[x2,y2]**...

CURVES([[x11,y11],...,[x1n,y1n]],[[x21,y21],...,[x2n,y2n]],...,[[xm1,ym1],...,[xmn,ymn]]) – **m** наборов по **n** точек, определяющих незамкнутые кривые. Точки каждого из наборов будут последовательно соединены отрезками.

POLYGONS([[x11,y11],...,[x1n,y1n]],[[x21,y21],...,[x2n,y2n]],...,[[xm1,ym1],...[xmn,ymn]]) – **m** наборов по **n** точек, определяющих замкнутые кривые. Точки каждого из наборов будут последовательно соединены отрезками, причем последняя точка набора соединяется с первой.

TEXT([x,y],`string`,horizontal,vertical) – графический вывод текста, начиная с позиции, определяемой координатами **[x,y]**. Опция **horizontal** определяет вид горизонтального выравнивания и может принимать одно из значений: **ALIGNLEFT** или **ALIGNRIGHT**. Способ вертикального выравнивания задает параметр **vertical** (**ALIGNABOVE** или **ALIGNBELOW**).

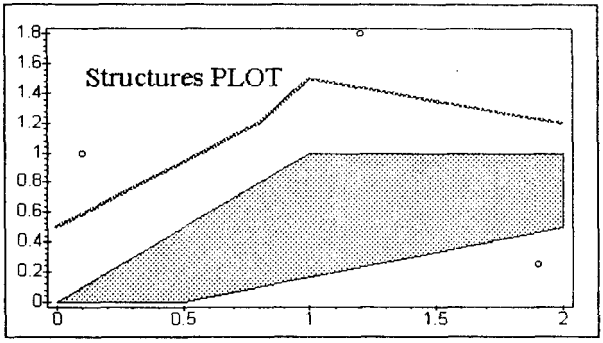
Для вывода графических структур **str1,...,strn** на экран используется команда стандартной библиотеки

```
PLOT({str1,...,strn});
```

При задании графических структур используются опции, аналогичные перечисленным выше, но отличающиеся синтаксисом. Некоторые примеры задания этих опций приведены ниже; при необходимости пользователь может обратиться к справке **?PLOT**.

Приведем пример, в котором использованы все двумерные структуры.

```
> var1:=POLYGONS([[0,0],[1,1],[2,1],
>     [2,1/2],[1/2,0]],COLOUR(RGB,1.0,1.0,0.0)):
> var2:=POINTS([1/10,1],[1.2,1.8],[1.9,1/4],
>       SYMBOL(CIRCLE)):
> var3:=TEXT([0.5,1.5],`Structures PLOT`,
>       FONT(TIMES,ROMAN,16)):
> var4:=CURVES([[0,0.5],[0.8,1.2],[1.0,1.5],
>           [2,1.2]],THICKNESS(2),COLOR(HUE,0)):
> PLOT(var1,var2,var3,var4,AXESSTYLE(BOX));
```



9.4. Опции трехмерной графики

В командах трехмерной графики для управления видом рисунка используются опции, частично совпадающие с опциями двумерной графики. Коротко перечислим основные опции трехмерной графики, не останавливаясь на уже описанных выше. По-прежнему установки по умолчанию отмечены в тексте подчеркиванием.

coords=opt – задание типа используемой системы координат (**CARTESIAN**, **SPHERICAL** или **CYLINDRICAL**),

title=string – вывод заголовка, содержащегося в строке **string**,

axes=opt – задание типа осей координат (**FRAME**, **NORMAL**, **BOXED**, **NONE**),

scaling=opt – задание типа масштабирования (**UNCONSTRAINED**, **CONSTRAINED**),

orientation=[angle1,angle2] – задание ракурса; углы **angle1,angle2** даются в градусах,

view=az..bz или **view=[ax..bx,ay..by,az..bz]** – определение выводимой на рисунок области (все за ее пределами отсекается),

projection=n – задание типа проекции (перспектива), $n \in [0,1]$,

style=opt – задание стиля вывода (**POINT** – точки, **LINE** – линии, **HIDDEN** – сетка с удалением невидимых линий, **PATCH** – заполнитель, **WIREFRAME** – сетка с выводом невидимых линий, **CONTOUR** – линии уровня, **PATCHCONTOUR** – заполнитель и линии уровня),

shading=opt – задание функции интенсивности заполнителя; параметр opt может принимать следующие значения: **Z** (функция координаты Z), **XY** (функция координат X и Y), **XYZ** (функция трех координат), **ZGREYSCALE**, **ZHUE**, **NONE** (без раскраски),

grid=[m,n] – задание числа узлов по осям X и Y для вычисления поверхности (**grid=[25,25]**),

numpoints=n – задание числа узлов для вычисления поверхности; эта опция эквивалентна опции **grid**=[$\sqrt{n}$,$\sqrt{n}$],

color=opt – задание цвета поверхности (**red,blue,..**),

light=[angl1, angl2, numr, numg, numb] – задание подсветки, создаваемой источником света из точки со сферическими координатами **angl1**, **angl2**. Цвет определяется долями красного (**numr**), зеленого (**numg**) и синего (**numb**) цветов, которые находятся в интервале от 0 до 1,

ambientlight=[numr,numg,numb] – задание цветности изображения. Цвет определяется долями красного (**numr**), зеленого (**numg**) и синего (**numb**) цветов, которые находятся в интервале от 0 до 1,

tickmarks=[i1, i2, i3] – задание числа насечек по осям координат,

labels=[str1,str2,str3] – надписи по осям координат, задаваемые строками **str1**,**str2**,**str3**.

Пользователь может переопределять установки на сеанс при помощи команды **setoptions3d(opt1=val1,..., optn=valn)**.

9.5. Структуры трехмерной графики

Для создания трехмерных изображений можно использовать как специальные графические структуры, так и основанные на них команды пакета **plots**. Структуры **POINTS**, **CURVES**, **POLYGONS**, **TEXT** аналогичны соответствующим двумерным структурам (см. выше) и имеют на одну размерность больше при задании координат точек. Опишем ниже специфически трехмерные графические структуры.

GRID(a..b,c..d,[[z11,...z1n],[z21,...z2n],...,[zm1...zmn]]) – задание поверхности над участком координатной плоскости [**a**,**b**]×[**c**,**d**] при помощи переменной типа **listlist**. Размерности этой переменной **n** и **m** определяют число равноотстоя-

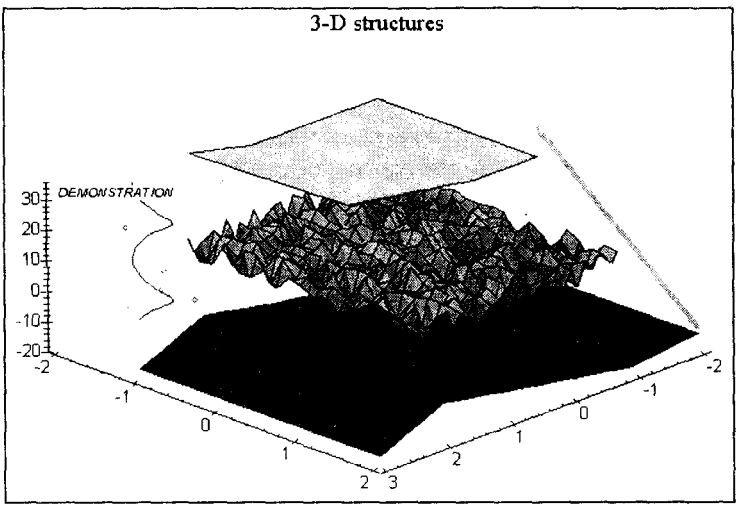
щих узлов по осям X и Y соответственно. Каждый элемент переменной `[[z11,...z1n],[z21,...z2n], ...,[zm1...zmn]]` задает координату Z в соответствующем узле.

`MESH([[[x11,y11,z11],...,[x1n,y1n,z1n]],[[x21,y21,z21],...,[x2n,y2n,z2n]],...,[[xm1,ym1,zm1]...[xmn,ymn,zmn]]])` – задание поверхности при помощи переменной типа `listlist`. В отличие от предыдущей команды здесь указываются все три координаты точек поверхности, что позволяет использовать неравномерную сетку.

Большинство опций являются общими для структур двумерной и трехмерной графики, и их имена близки к опциям графических команд. Продемонстрируем использование всех структур трехмерной графики и применение некоторых опций при помощи следующего примера. Вначале командой пакета линейной алгебры генерируется матрица со случайными элементами и преобразуется в переменную A типа `listlist`, затем формируются структуры `GRID`, `MESH` и `POLYGONS`, задающие три поверхности. Далее определяются шесть точек (структура `POINTS`), пространственные кривая и прямая (структуры `CURVES`) и текст (структура `TEXT`). Все эти структуры при помощи команды `PLOT3D` выводятся на одном рисунке, при этом указываются дополнительные опции.

```
> A:=convert(evalm(linalg[randmatrix](30,20)/20)
>            ,listlist):
> pic1:=GRID(-2..1,-2..1,A):
> pic2:=MESH([[[-2,-1,-14],[-2,2,-14]],[[-1,-1,
>      -17],[-1,2,-17]],[[1,-1,-10],[1,2,-10]],
>  [[2,-1,-6],[2,2,-6]],[[3,-1,-16],[3,2,-16]]],
>     GRIDSTYLE(TRIANGULAR)):
> pic3:=POLYGONS([[-1,0,25],[-2,0,25],
>   [-2,-2,25],[0,-2,25],[1,-2,30],[1,-0,32]]):
> pic4:=POINTS([2,-2,-13],[1.8,-1.7,-6],
>              [1.9,-1.2,-3.5],[2,-1.5,4],
>              [2.1,-1.9,15],[1.7,-1,35],
>               SYMBOL(DIAMOND),COLOR(HUE,3.7)):
> pic5:=CURVES([seq([evalf(1.8+0.2*sin(x/4)),
>      -1.8+0.2*evalf(cos(x/4)),x-7],x=-12..35)],
>         COLOR(RGB,0.1,0.1,0.1)):
> pic6:=CURVES([[-2,2,-12],[-2,0,35]],
```

```
>                THICKNESS(3)):
> pic7:=TEXT([3,-2,30],`DEMONSTRATION`,
>        ALIGNRIGHT,FONT(HELVETICA,OBLIQUE,8),
>        ALIGNABOVE,COLOR(RGB,0.5,0.5,0.5)):
> PLOT3D( pic1,pic2,pic3,pic4,pic5,pic6,pic7,
>        AXES(FRAME),TITLE(`3-D structures`),
>        STYLE(PATCH),COLOUR(ZHUE));
```



9.6. Команды трехмерной графики

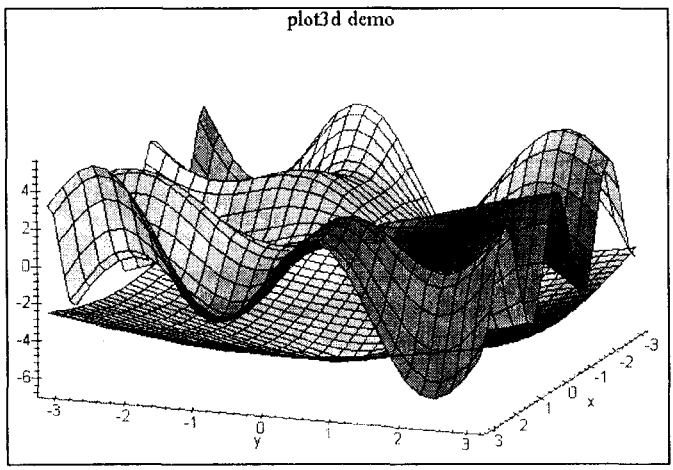
В данном разделе приводятся наиболее популярные команды трехмерной графики. Многие из них аналогичны командам двумерной графики и имеют имена с окончаниями '3d'. При этом число параметров, как правило, больше на единицу, а точки определяются тремя координатами. Как и в случае двумерной графики, где функция `plot` выполняет самые разнообразные функции, в трехмерном случае существует многофункциональная команда:

```
plot3d({ex1,ex2,...},var1=a..b,var2=c..d,<options>).
```

Здесь приведен общий вид команды, позволяющей выводить на одном рисунке несколько поверхностей, задаваемых однотипными Maple-выражениями **ex1**, **ex2**,... . Эти выражения должны зависеть от двух переменных **var1** и **var2**. Поверхности рисуются для прямо-

угольника **var1**∈[**a**,**b**], **var2**∈[**c**,**d**]. Вид представления рисунка можно менять при помощи опций (**options**). Приведем пример использования этой команды для изображения двух поверхностей:

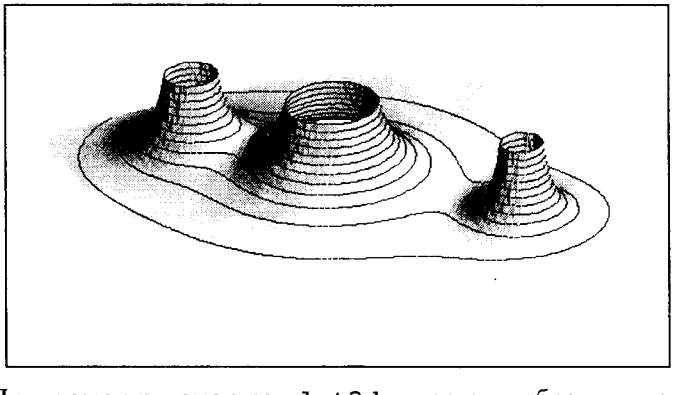
```
> plot3d({x*sin(2*y)+y*cos(3*x),sqrt(x^2+y^2)-7},
>     x=-Pi..Pi,y=-Pi..Pi,
>     grid=[30,30],title=`plot3d demo`,
>     axes=FRAMED,orientation=[20,65],color=x+y);
```



При построении сложных трехмерных изображений полезно использовать возможности меню графики Windows-версии для интерактивного подбора значений некоторых опций. Так, управляя мышью, можно подобрать лучший ракурс (опция **orientation**), а при помощи пунктов меню определить оптимальные значения опций (**style**, **shading**, **axes** и т.д.). Эти назначения затем можно вставить в команду. Однако ряд опций можно задать, только явно указывая их при вызове команды. Например, такими опциями являются **title**, **view**, **light**. Ниже представлен пример команды **plot3d**, в котором при помощи этих опций определяются область вывода (**view**) и подсветка (**light**).

```
> plot3d( 1/(x^2+y^2)+0.2/((x+1.2)^2+(y-1.5)^2)+
>      0.3/((x-0.9)^2+(y+1.1)^2), x=-2..2,y=-2..2.5,
>     view=[-2..2,-2..2.5,0..6],
>     grid=[60,60],shading=NONE,
>     light=[100,30,1,1,1],axes=NONE,
```

```
> orientation=[65,20],style=PATCHCONTOUR);
```



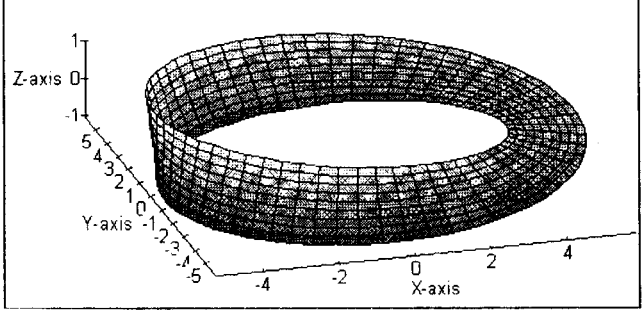
При помощи команды **plot3d** можно изобразить также параметрически заданную поверхность:

plot3d([expr1,expr2,expr3],var1=a..b,var2=c..d),

где координаты заданы выражениями **expr1**,**expr2**,**expr3** от двух переменных **var1** и **var2**, которые изменяются на отрезках [**a**,**b**] и [**c**,**d**] соответственно.

Например:

```
> plot3d([(5+cos(t/2)*u)*cos(t),
>  (5+cos(t/2)*u)*sin(t),sin(t/2)*u], t=0..2*Pi,
>    u=-1..1, grid=[60,10], scaling=CONSTRAINED,
>    orientation =[-106,70],
>    title=`Moebius band.`, axes = FRAMED,
>    shading=ZGREYSCALE, tickmarks= [5,8,3],
>    labels=[`X-axis`,`Y-axis`,`Z-axis`]);
```



При помощи команды **plot3d** можно осуществлять и некоторые другие операции (см. Справку по работе с Maple).

Кроме функции **plot3d** в программе Maple имеются следующие процедуры трехмерной графики:

contourplot(expr,var1=a..b,var2=c..d,<options>) – построение линий уровня функции **expr** двух переменных **var1** и **var2** в прямоугольнике [a,b]×[c,d] (идентична команде **plot3d** с опцией **style=CONTOUR**).

cylinderplot(expr,var1=a..b,var2=c..d,<options>) – построение поверхности, определяемой выражением **expr**, заданным в цилиндрических координатах.

fieldplot3d([expr1,expr2,expr3], var1=a..b, var2=c..d, var3=e..f, <options>) – построение в параллелепипеде [a,b]×[c,d]×[e,f] трехмерного векторного поля с компонентами, заданными выражениями **expr1, expr2, expr3**, которые зависят от переменных **var1**, **var2**, **var3**.

gradplot3d(expr,var1=a..b,var2=c..d,var3=e..f,<options>) – изображение трехмерного поля градиента выражения **expr**, зависящего от трех переменных **var1**, **var2**, **var3**. Переменные изменяются в заданных интервалах.

implicitplot3d(expr=g,var1=a..b,var2=c..d,var3=e..f, <options>) – построение неявно заданной поверхности **expr=g** в параллелепипеде [a,b]×[c,d]×[e,f]. Здесь выражение **expr** зависит от переменных **var1**, **var2**, **var3**.

matrixplot(M,<options>) – построение поверхности, заданной матрицей **M**. При задании опции **heights=histogram** строится трехмерная гистограмма.

odeplot(s,vars,r1,r2,<options>) – изображение решения дифференциального уравнения (см. примеры в гл. 10 "Мини-исследования" и гл. 6 "Дифференциальные уравнения").

pointplot([[x1,y1,z1],...,[xn,yn,zn]],<options>) – вывод точек, заданных своими координатами.

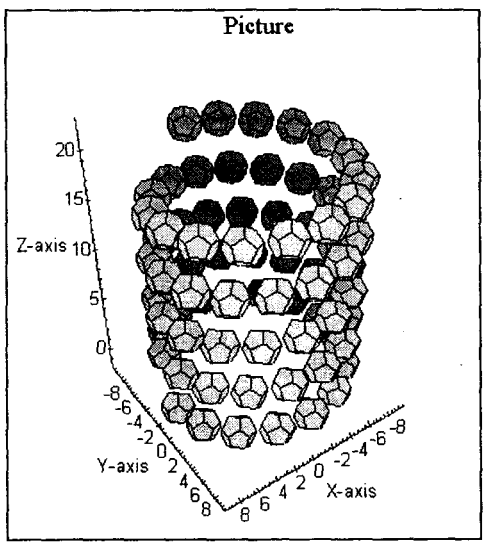
polygonplot3d([pnt1,...,pntn]) – построение трехмерного n-угольника, заданного точками **pnt1,...,pntn**, причем первая точка соединяется с последней.

polyhedraplot(L,<options>) – изображение набора точек в виде трехмерных геометрических фигур. Здесь **L** – переменная типа **listlist**, в которой находятся координаты точек. Каждая точка является центром изображаемой фигуры. Опция **polyscale=n**

задает размер фигуры (по умолчанию n=1), а опция **polytype=set** определяет вид фигуры, где **set** может принимать следующие значения: **tetrahedron**, **octahedron**, **hexahedron**, **dodecahedron**, **icosahedron**.

Приводимый пример состоит из двух команд. Сначала при помощи команды **seq** формируется последовательность точек, задающая спираль. Затем командой **polyhedraplot** эта последовательность выводится так, что каждая точка превращается в додекаэдр. Отметим, что здесь использована опция **projection** для определения близкой перспективы и опция **scaling** для задания масштаба.

```
> p:=seq([8*cos(t/2.5),
>        8*sin(t/2.5),t/3.5],t=0..75):
> polyhedraplot([p], polyscale=1.4,
>   polytype= dodecahedron, scaling=CONSTRAINED,
>   orientation=[55,55], title=`Picture`,
>   projection=1/10, labels=[`X-axis`,`Y-axis`,
>   `Z-axis`],axes=FRAMED);
```



spacecurve([expx,expy,expz],var=a..b,<options>) – построение кривой в трехмерном пространстве. Кривая задается параметрически выражениями **expx**, **expy** и **expz** (X, Y и Z –коорди-

наты), зависящими от одной переменной **var**, изменяющейся на интервале [**a**,**b**].

sphereplot(expr,var1=a..b,var2=c..d,<options>) – построение поверхности, заданной в сферических координатах. Выражение **expr** задает радиус, зависящий от двух угловых переменных **var1** и **var2**.

surfdata([[[x11,y11,z11],...,[x1n,y1n,z1n]],[[x21,y21,z21],...,[x2n,y2n,z2n]],...,[[xm1,ym1,zm1]...[xmn,ymn,zmn]]],<options>) – построение поверхности, заданной переменной типа **listlist**.

textplot3d([expx,expy,expz],string,<options>) – вывод текстовой строки **string** с центром в позиции, задаваемой координатами **expx**, **expy** и **expz**.

tubeplot([exprx,expry,exprz],var=a..b,,radius=r,<options>) – построение поверхности вращения, полученной обращением радиуса **r** вокруг параметрически заданной пространственной кривой. Кривая определяется выражениями **exprx**, **expry** и **exprz** (X, Y и Z –координаты), зависящими от одной переменной **var**, изменяющейся на интервале [**a**,**b**].

setoptions3d(options1=exp1,...,optionsn=expn) – назначение опций трехмерной графики на сеанс.

display3d([pic1,...,picn],<options>) – вывод трехмерных графических образов **pic1**,...,**picn** в общем масштабе. Специфические опции этой команды аналогичны соответствующим опциям двумерной команды **display**.

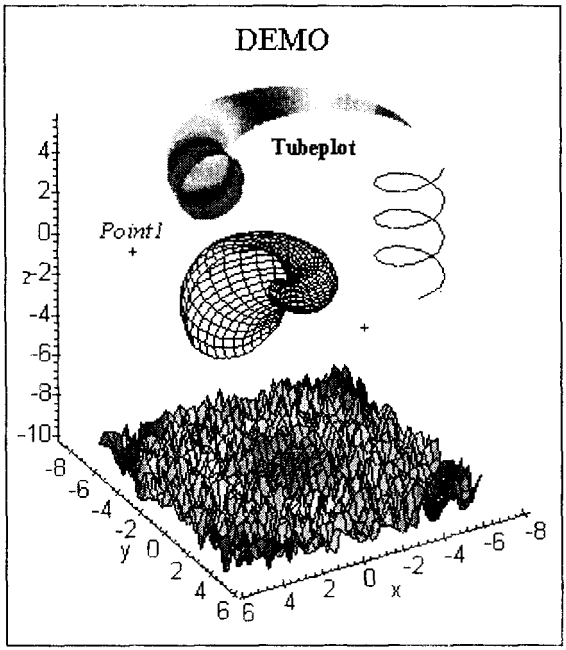
Коротко коснемся возможностей трехмерной мультипликации. Один способ заключается в использовании команды **display3d** с использованием опции **insequence=true**, а другой использует команду:

animate3d(F, x=a..b, y=c..d, t=t1..t2, frame=n, <options>) - здесь **F** – выражение, зависящее от переменных **x**,**y** и **t**, которые изменяются соответственно в интервалах [**a**,**b**], [**c**,**d**] и [**t1**,**t2**]. Переменные **x** и **y** отвечают оси абсцисс и ординат соответственно, **t** – переменная времени, а опция **frame** задает число кадров.

Приведем пример, демонстрирующий работу некоторых команд трехмерной графики. При помощи команды **display3d** на од-

ном рисунке изображены построенная с использованием сферических координат поверхность (*pic1*), поверхность вращения (*pic4*) и заданная случайной матрицей поверхность (*pic6*). Здесь же представлена пространственная кривая (*pic5*), две помеченные крестиками точки (*pic2*), и нанесены надписи (*pic3* и *pic7*). Отметим, что для раскраски двух поверхностей (*pic4* и *pic6*) использованы специально введенные функции цвета (*F* и *f1* соответственно).

```
> with(plots):
> pic1:=sphereplot((1.3)^z*sin(theta),
>        z=-1..2*Pi, theta=0..Pi,
>        style=HIDDEN, shading=NONE):

> pic2:=pointplot([[6,-3,2],[-2,3,-1]],
>             color=blue,symbol=CROSS):
> pic3:=textplot3d([6,-3,3.0,`Point1`],
>         color=black, font=[TIMES,ITALIC,10]):

> F:=(x,y)->sin(x):
> pic4:=tubeplot([-3+5*cos(t),-3+5*sin(t),4,
>     t=Pi..2*Pi,radius=0.5*(t-Pi)],
>     tubepoints=18,
>     color=F,style=PATCHNOGRID):

> pic5:=spacecurve([-6+1.5*cos(t/2),1.5*sin(t/2)
>                ,t/7], t=-10..30,color=black):

> f1:=(x,y)->(x^2+y^2):
> dd:=rand(0..20):
> A:=seq([seq([i/3,j/3,-10+evalf(dd()/10)],
>          i=-18..18)], j=-18..18):

> pic6:=surfdata([A],color=f1):

> pic7:=textplot3d([-3,-3,4.5,`Tubeplot`],
>         color=black,font=[TIMES,BOLD,9]):

> display3d([pic1,pic2,pic3,pic4,pic5,pic6,pic7]
>           ,axes=FRAMED,orientation=[60,60],
>           scaling=CONSTRAINED,title=`DEMO`);
```



9.7. Иллюстративные графические команды

В предыдущих разделах речь шла о командах общего назначения графического пакета `plots`. В состав других пакетов Maple также входят графические команды. Обычно они предназначены для иллюстрирования результатов работы математических команд соответствующих пакетов. Опишем некоторые специализированные графические команды, начав с пакета `student`.

Для иллюстрации численного интегрирования функции `f(x)` на интервале `[a,b]` при помощи разбиения на n прямоугольников имеются следующие команды:

`leftbox(f(x),x=a..b,n,<options>)` – метод левых прямоугольников.

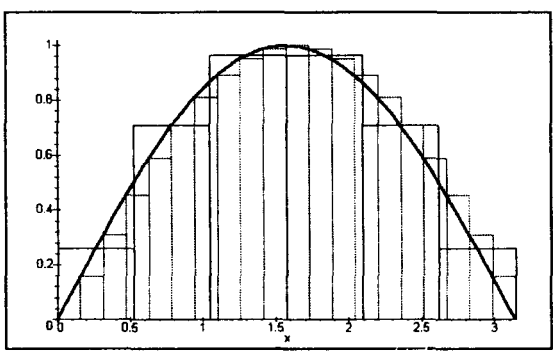
`rightbox(f(x),x=a..b,n,<options>)` – метод правых прямоугольников.

`middlebox(f(x),x=a..b,n,<options>)` – метод центральных прямоугольников.

Для построения касательной к функции **f(x)** в точке **a** используется команда **showtangent(f(x),x=a,<options>)**.

Приведем примеры:

```
> p1:=student[leftbox](sin(x),x=0..Pi,20):
> p2:=student[middlebox](sin(x),x=0..Pi,6):
> p3:=plot(sin(x),x=0..Pi,thickness=2):
> plots[display]({p1,p2,p3});
```



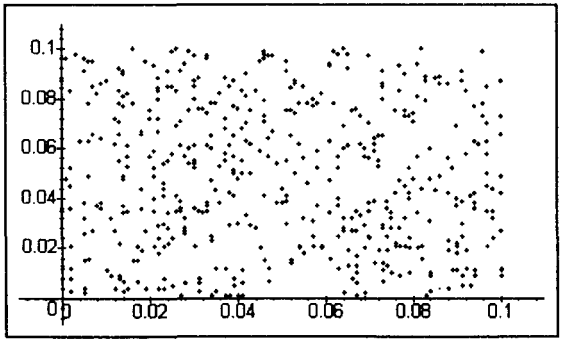
В статистическом пакете **stats** есть целая подбиблиотека **statplots** иллюстративных графических команд, приведем две из них:

histogram(xdata) – построение гистограммы для набора данных **xdata** типа **list**,

scatter2d(xdata,ydata) – изображение точек с абсциссами **xdata** и ординатами **ydata**.

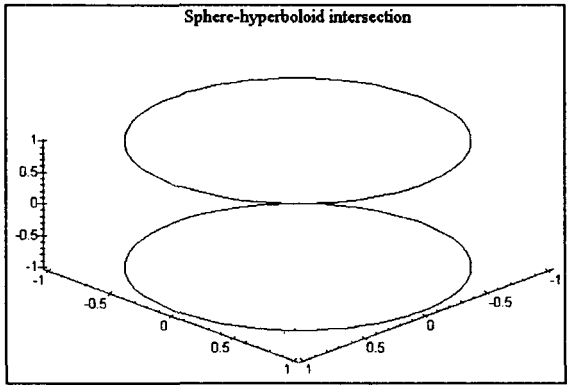
Приводимый пример вывода последовательности точек показывает также, как в Maple генерировать наборы случайных чисел при помощи команды **seq**.

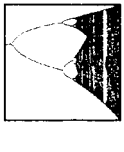
```
> with(stats): dd:=rand(0..1000);
> xdata:=[seq(evalf(dd()/100),i=1..200)];
> ydata:=[seq(evalf(dd()/100),i=1..200)];
> statplots[scatter2d](xdata,ydata);
```



В заключение этой главы приведем пример использования одной команды из библиотеки **share**. Эта команда предназначена для изображения пересечения двух поверхностей.

```
> with(share);
> readshare(intplot, plots);
                              intplot
> intersectplot(x^2+y^2-1,x^2+y^2+z^2-2,
> [x,y,z],shading=ZHUE,axes=FRAMED,
> title=`Sphere-hyperboloid intersection`);
```





10. Мини-исследования

В данной главе рассмотрен ряд мини-исследований – выполненных средствами Maple законченных работ по решению интересных проблем. В качестве тем выбраны известные объекты теории динамических систем – логистическое отображение и система Лоренца, физическая задача о стационарном течении вязкой жидкости в канале и математическая задача разложения функции в ряд Фурье.

Назначение этих примеров – показать возможности Maple для решения сложных проблем и попутно проиллюстрировать действие возможно большего числа команд; при этом не всегда использование той или иной команды объясняется ее необходимостью, а иногда вызвано и нашей попыткой объять необъятное.

10.1. Логистическое отображение

Данный пример посвящен исследованию логистического отображения – очень популярного объекта для демонстрации хаоса в динамических системах [5,6]. Эта задача была в числе первых двадцати задач, которые были предложены для решения на ЭВМ, частично освобожденной от расчета атомной бомбы в США. Логистическое отображение является прекрасным примером для изучения каскада удвоений – последовательности бифуркаций рождения периодических режимов, ведущей к хаосу.

Изучим возможности явного определения периодических режимов и вычисления их устойчивости, проверим способность Maple рассчитывать итерации отображения и др. Наконец, при помощи данного примера продемонстрируем следующие возможности Maple: задание функций, символьное дифференцирование и интегрирование, аналитическое и численное определение корней полиномов, решение уравнений, программирование процедур, операторов цикла и условных выражений, использование двумерной графики.

Запишем логистическое отображение в виде

$$x_{n+1} \to f(x_n) = bx_n(1-x_n), \qquad (1)$$

здесь $x \in [0,1]$, $b \in [0,4]$.

Неподвижные точки отображения (1) удовлетворяют уравнению $x = f(x)$. Циклом периода p назовем такую последовательность точек $\{x_1, x_2, \ldots, x_p\}$, что

$$x_2 = f(x_1), \ldots, x_p = f(x_{p-1}), x_1 = f(x_p).$$

При этом для отображения $x \to f^p(x) = f(f(\ldots f(x)\ldots))$ каждая из точек $x_1, x_2, \ldots, x_p$ является неподвижной:

$$x_k = f^p(x_k) = f(f(\ldots f(x_k)\ldots)),$$

Среди решений $x_k = f^p(x_k)$ кроме точек, составляющих цикл периода p, будут также находиться неподвижные точки периодов, которые выступают делителями числа p.

Цикл называется устойчивым, если значение мультипликатора по модулю меньше единицы. Для одномерных отображений мультипликатор рассчитывается очень просто: это произведение производных во всех точках, составляющих цикл

$$\prod_{k=1}^{p} f'(x_k).$$

Далее приведем фрагменты из программы, предваряя их краткими пояснениями.

Вначале зададим само отображение, используя специальную конструкцию, и вычислим производную правой части отображения, применив команду `unapply`, поскольку новая функция определяется через известную. Заметим, что по ходу задания новой функции использована команда `factor` для выделения сомножителей. Ясно, что хорошая организация выражения (упрощение, факторизация и др.) ускоряет последующую работу с ним, да и формулы легче читаются.

```
> f:=x->b*x*(1-x);
```

$$f := x \to b\,x\,(1-x)$$

```
> df:=unapply(-factor(D(f)(x)),x);
```

$$df := x \to b\,(-1+2\,x)$$

Найдем предельное значение параметра b, при котором отрезок $[0,1]$ отображается в себя. Здесь можно обратить внимание на то, что команды вкладываются друг в друга, и результат решения одного уравнения участвует в задании другого уравнения.

```
> b_lim:=solve( 1=f( solve(df(x)=0,x) ) , b );
```

$$b\_lim := 4$$

Далее вычислим неподвижные точки квадрата отображения при помощи команды **solve**, результаты поместим в переменную **fixed** и проверим прямой подстановкой, что вторая из найденных точек является неподвижной точкой отображения. Отметим, что результат получен в виде некоторого выражения, где фигурирует параметр b. Вычислим значение мультипликатора для неподвижной точки, и решим неравенство, $|f'(x)|<1$, из которого найдем интервал устойчивости неподвижной точки.

```
> fixed:=solve(x=f(f(x)),x);
```

$$fixed := 0, -\frac{1-b}{b}, \frac{1}{2}\frac{b^2+b+\sqrt{b^4-2b^3-3b^2}}{b^2}, \frac{1}{2}\frac{b^2+b-\sqrt{b^4-2b^3-3b^2}}{b^2}$$

```
> simplify( f (fixed[2]) )=simplify( fixed[2] );
```

$$\frac{-1+b}{b} = \frac{-1+b}{b}$$

```
> r_f:=simplify(df(fixed[2]));
```

$$r\_f := b-2$$

```
> solve(abs(r_f)<1,b);
```

$$\{1<b, b<3\}$$

Нетрудно убедиться, что третье и четвертое значения из **fixed** дают цикл периода два для исходного отображения.

Определить область устойчивости цикла периода два можно аналогично. Вычислим мультипликатор и решим соответствующее неравенство. После этого выделим подходящее решение

(отрицательные значения параметра b не рассматриваются) и преобразуем ответ, содержащий радикал, в вещественное число.

```
> bb:=expand( df(fixed[3]) * df (fixed[4]) );
```

$$bb := 4 - b^2 + 2\,b$$

```
> stab_2:=solve(abs(bb)<1,b);
```

$$stab\_2 := \left\{1 - \sqrt{6} < b, b < -1\right\}, \left\{3 < b, b < 1 + \sqrt{6}\right\}$$

```
> ss:=rhs(convert(stab_2[2][2],equality));
> evalf(ss);
```

$$ss := 1 + \sqrt{6}$$

$$3.449489743$$

Далее определим куб отображения и его производную.

```
> f3:=unapply(f(f(f(x))),x);
```

$$f3 := x \to b^3 x (1-x)(1 - b\,x(1-x))\left(1 - b^2 x (1-x)(1 - b\,x(1-x))\right)$$

```
> df3:=unapply(-factor(D(f3)(x)),x);
```

$$df3 := x \to b^3 (-1 + 2\,x)\left(2\,b\,x^2 - 2\,b\,x + 1\right)\left(2\,b^3 x^4 - 4\,b^3 x^3 + 2\,b^3 x^2 + 2\,b^2 x^2 - 2\,b^2 x + 1\right)$$

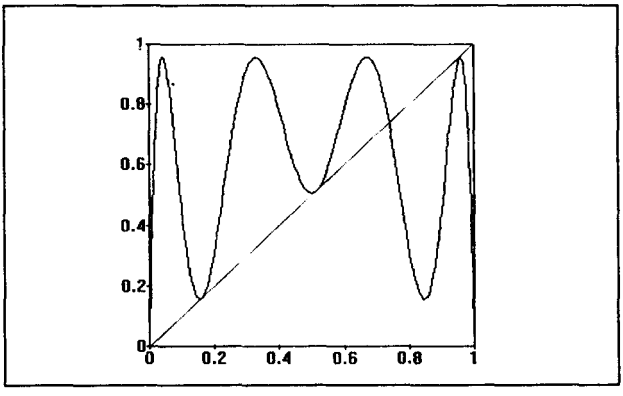
На примере цикла периода три покажем, как использовать график для поиска циклов различных периодов. Для этого определим прямую $y = x$, пересечения с которой графика куба отображения укажут на существование нужного цикла. Для рисования требуется определенность всех переменных (кроме x), поэтому нужно задать конкретное значение параметра b.

```
> bis:=x->x; b:=3.828428;
```

$$bis := x \to x$$

$$b := 3.828428$$

```
> plot({f3,bis},0..1,0..1,axes=BOXED,
>      scaling=constrained);
```



Когда на графике имеются три парных пересечения прямой $y = x$, то это говорит о наличии цикла периода три. Для численного поиска точек цикла можно обратиться к команде **fsolve**. По рисунку, примерно представив тот интервал, где располагается пара корней, зададим его, упростив задачу нахождения решений. Затем вычислим все точки циклов и выясним устойчивость самих циклов. Используя возможность Maple устанавливать любую длину мантиссы, меняя значение переменной **Digits**, проверим результат, увеличив количество значащих цифр.

```
> fff:=fsolve(f3(x)=x,x,0.1..0.3);
```

$$fff := .1598419562, .1600156014$$

```
> Digits:=25;  fsolve(f3(x)=x,x,0.1..0.3);
```

$$Digits := 25$$

$$.15984118656555579342623021,$$
$$.16001637325362280899291 47$$

```
> Digits:=10;
```

$$Digits := 10$$

```
> df3(fff[1]); df3(fff[2]);
```

$$-.9845561276$$
$$-1.015433828$$

Конечно, являясь системой символьных вычислений, Maple уступает в скорости специализированным программам расчета динамических систем для итерирования отображений. Однако Maple пре-

доставляет возможность сочетания аналитических и численных методов исследования. Приводимая процедура показывает, что для вычисления итераций отображения можно воспользоваться паскалеподобным языком программирования Maple. Следует обратить внимание на команду **convert** для преобразования данных к формату, удобному для дальнейшего построения графика.

```
> Iterates := proc(n,x)
> local It,i; It:= array(-1..2*n);
> if n<0 or x<0 or x>1 then
>    ERROR(`invalid argument`) fi;
> It[-1]:=0; It[0]:=x;
> for i from 1 to n do It[2*i-1]:=i;
> It[i*2]:=f(It[2*i-2]) od;
> RETURN(convert(It,list)) end;
```

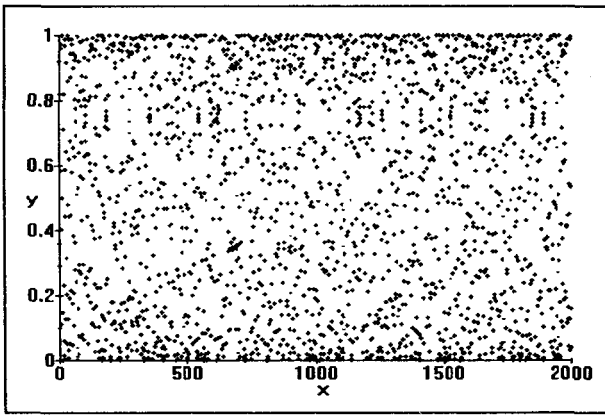
```
Iterates := proc(n,x)
    local It,i;
      It := array(-1 .. 2*n);
      if n < 0 or x < 0 or 1 < x then
        ERROR(`invalid argument `)
      fi;
      It[-1] := 0;
      It[0] := x;
      for i to n do
        It[2*i-1] := i; It[2*i] := f(It[2*i-2])
      od;
      RETURN(convert(It,list))
    end
```

Результат работы процедуры в виде графика зависимости переменной x от номера итерации для предельного значения параметра b приведен ниже. Видно, что поведение системы хаотично.

```
> b:=4;
```

$$b := 4$$

```
> n:=2000;
> plot(Iterates(n,0.1),x=0..n,y=0..1,style=POINT);
```



$n := 2000$

Важной при изучении хаоса [6] в динамической системе бывает информация об инвариантной мере или плотности распределения точек на отрезке. Следующий фрагмент показывает, что экспериментальное распределение точек по отрезку можно найти, написав небольшую процедуру со следующими параметрами: n – число итераций отображения из начальной точки x, а m – число разбиений единичного отрезка на интервалы. Результатом работы процедуры является массив частоты попадания точек в заданные интервалы.

```
> Density := proc(n,m,x) local It,count,i,k;
> It:=array(-1..2*n); count:= array(1..2*m);
> if n<0 or x<0 or x>1 or m<1 then
>    ERROR(`invalid argument `)
> fi; It:=Iterates(n,x);
> for i from 1 to m do count[2*i-1]:=(i-1/2)/m;
> count[2*i]:=0;od;
> for i from 1 to n do k:=2+2*floor(m*It[i*2]);
> count[k]:=count[k]+m/n; od;
> RETURN(convert(count,list)) end;
```

```
Density :=
  proc(n,m,x)
  local It,count,i,k;
    It := array(-1 .. 2*n);
    count := array(1 .. 2*m);
```

```
    if n < 0 or x < 0 or 1 < x or m < 1 then
      ERROR(`invalid argument `)
    fi;
    It := Iterates(n,x);
    for i to m do
      count[2*i-1] := (i-1/2)/m; count[2*i] := 0
    od;
    for i to n do
      k := 2+2*floor(m*It[2*i]);
      count[k] := count[k]+m/n
    od;
    RETURN(convert(count,list))
  end
```

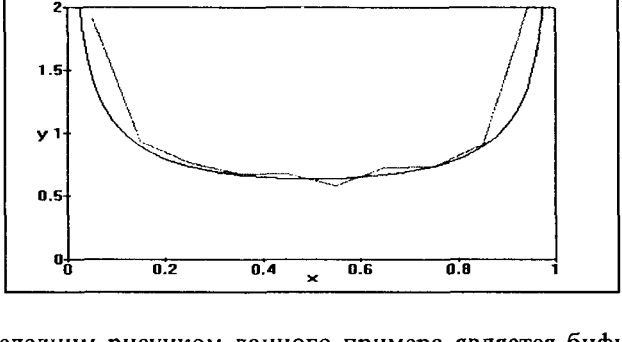
Рисунок же показывает, что на графике легко соседствуют таблично заданная функция плотности и имеющееся для этого значения параметра b аналитическое выражение для инвариантной плотности, даже с особенностями на концах интервала.

```
> rho:=1/Pi/sqrt(x*(1-x)); int(rho,x=0..1);
```

$$\rho := \frac{1}{\pi\sqrt{x}\sqrt{1-x}}$$

$$1$$

```
> plot({rho,Density(2000,10,.1)},x=0..1,y=0..2);
```



Последним рисунком данного примера является бифуркационная диаграмма, показывающая, какие аттракторы реализуются при изменения параметра задачи. Алгоритм расчета диаграммы прост: для конкретного значения параметра b проводится расчет на уста-

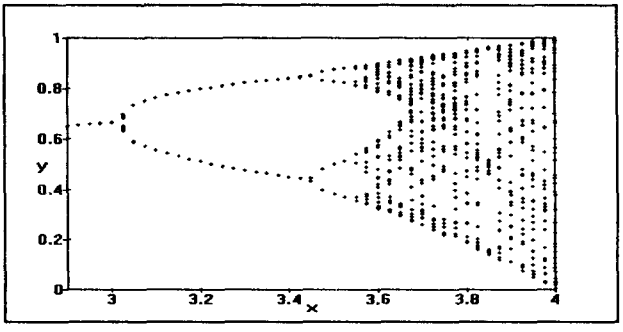
новление, а затем совершается некоторое количество итераций, способных дать представление о соответствующем аттракторе. Понятно, что для циклов малых периодов не нужно столько точек, сколько требуется для изображения хаотических режимов. Приводимый рисунок получен при компромиссных назначениях числа точек на отрезке изменения параметра и числа точек, отводимых на изображение аттрактора.

Параметрами процедуры являются *nn* – число разбиений интервала изменения [*bbeg,bend*] параметра *b*, *pre* – число итераций при расчете на установление, *all* – число точек для представления аттрактора.

```
> Diagram := proc(bbeg,bend,nn,pre,all)
> local x0, i, n, b,db; It:= array(1..2*nn*all);
> x0:=1-1/bbeg; db:=(bend-bbeg)/(nn-1) ; n:=0;
> for b from bbeg by db to bend do
> for i from 1 to pre do x0:=b*x0*(1-x0) od;
> for i from 1 to all do n:=n+2; It[n-1]:=b;
> x0:=b*x0*(1-x0); It[n]:=x0 od; od;
> plot(convert(It,list),x=bbeg..bend,y=0..1,
> style=POINT,axes=BOXED,scaling=constrained);
> end;
```

```
Diagram :=
proc(bbeg,bend,nn,pre,all)
local x0,i,n,b,db,It;
  It := array(1 .. 2*nn*all);
  x0 := 1-1/bbeg;
  db := (bend-bbeg)/(nn-1);
  n := 0;
  for b from bbeg by db to bend do
    for i to pre do  x0 := b*x0*(1-x0) od;
    for i to all do
      n := n+2;
      It[n-1] := b;
      x0 := b*x0*(1-x0);
      It[n] := x0
    od
  od;
  plot(convert(It,list),x = bbeg .. bend,y = 0 .. 1,
```

```
    style = POINT,axes = BOXED,
    scaling = constrained)
end
> Diagram(2.9, 4, 55, 45, 45);
```



10.2. Разложение функции в ряд Фурье

В этом разделе мы предлагаем процедуру, которая для заданного выражения выписывает отрезок ряда Фурье. Входными параметрами процедуры будут: *f* – выражение, для которого нужно выписать ряд Фурье, *x* – переменная, *[-a,a]* – интервал изменения *x*, *n* – количество членов ряда. Ниже приведен текст процедуры с комментариями (область **Text Region**).

Заголовок процедуры: имя - fourieseries, параметры: f - разлагаемое выражение, отрезок по x [-a,a], n - число членов ряда. Описаны две локальные переменные i,s.

```
> fourieseries:=proc(f,x,a,n) local i,a,b,s;
```

Здесь мы проверяем, не является ли интервал [-a,a] нулевым. Если величина a=0, то при помощи процедуры ERROR выдается сообщение об ошибке.

```
> if a=0 then ERROR(`3 argument must be <> 0`);fi;
```

Проверка числа членов ряда: эта величина должна быть строго положительной.

```
> if n<0 then ERROR(`4 argument must be >=0`); fi;
```

Проверка типа выражения f, которое должно иметь тип algebraic.

```
> if (not type(f,`algebraic`))
> then ERROR(`1'st argument is not function`); fi;
```

Проверка типа переменной x, которая должна иметь тип string.

```
> if not type(x,`string`)
>       then ERROR(`2'st argument is not name
>                  of variable`); fi;
```

Переменной s присваивается нулевой член ряда Фурье.

```
> s:=int(f,x=-a..a)/l/2;
```

Вычисление суммы из n членов ряда Фурье.

```
> for i  from 1 to n do
>     s:=s+int(f*cos(i*Pi*x/a),x=-a..a)/a*
>          cos(i*Pi*x/a)+
>          int( f*sin( i*Pi*x/a),x=-a..a)/a*
>          sin(i*Pi*x/a);
> od;
```

Возвращение в качестве результата отрезка ряда Фурье.

```
> s; end:
```

Проверим теперь работу предложенной процедуры на примере из задачника Демидовича [4]. Выпишем первые шесть членов ряда Фурье, затем построим график самой функции и полученного разложения.

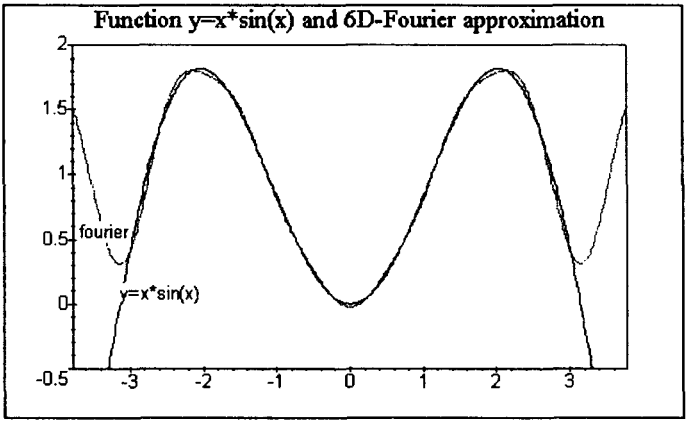
д2691. Разложить в ряд Фурье функцию $f = x\sin(x)$ на отрезке $[-\pi,\pi]$.

```
> f:=x*sin(x):
> g:=fourieseries(f,x,Pi,6);
```

$$g := 1 - \frac{1}{2}\cos(x) - \frac{2}{3}\cos(2x) + \frac{1}{4}\cos(3x) - \frac{2}{15}\cos(4x) + \frac{1}{12}\cos(5x) - \frac{2}{35}\cos(6x)$$

```
> pic1:=plot({f,g},x=-1.2*Pi..1.2*Pi,
>            -0.5..2,numpoints=500):
> pic2:=textplot([-1.07*Pi,
>            subs(x=-1.07*Pi,g),`fourier`]):
> pic3:=textplot([-Pi,0,`y=x*sin(x)`],
>                align={ABOVE,RIGHT}):
```

```
> with(plots):
> display({pic1,pic2,pic3},axes=BOXED,
>          title=`Function y=x*sin(x)
>          and Fourier approximation`);
```



10.3. Система Лоренца как маятник с обратной связью

В этом разделе рассматривается система трех обыкновенных дифференциальных уравнений первого порядка, которая описывает движения регулируемого маятника с одной степенью свободы. К этой системе при помощи замен сводится знаменитая система Лоренца (см., например, [5–8]). В этом примере средствами Maple иллюстрируются результаты работы В.И.Юдовича [8].

Система Лоренца имеет вид:

$$\begin{cases} \dot{u} = -\sigma u + \sigma v \\ \dot{v} = ru - v - uw \\ \dot{w} = uv - bw \end{cases} \qquad (2)$$

здесь σ, b, r – параметры, а u, v, w – неизвестные функции времени t. Известно, что при значении параметров σ=10, b=8/3, r=28 у системы существует странный аттрактор (решение задачи Коши обнаруживает сложное поведение, которое приближается по своим свойствам к стохастическому).

Как показано в [8], после замены переменных и введения новой независимой переменной систему (2) можно привести к следующему виду:

```
> eq1_0:=diff(x(t),t$2)+epsilon*h*diff(x(t),t)
>        +x(t)^3+(q(t)-1)*x(t)=0;
> eq2_0:=diff(q(t),t)=
>        -epsilon*a*q(t)+epsilon*beta*x(t)^2;
```

$$eq1\_0 := \left(\frac{\partial^2}{\partial t^2}\,x(t)\right) + \varepsilon\, h\left(\frac{\partial}{\partial t}\,x(t)\right) + x(t)^3 + (q(t)-1)\,x(t) = 0$$

$$eq2\_0 := \frac{\partial}{\partial t}\,q(t) = -\varepsilon\, a\, q(t) + \varepsilon\, \beta\, x(t)^2$$

Параметры этой системы дифференциальных уравнений связаны с параметрами исходной соотношениями:

$$\varepsilon = \frac{1}{\sqrt{r-1}},\quad h = \frac{\sigma+1}{\sqrt{\sigma}},\quad a = \frac{b}{\sqrt{\sigma}},\quad \beta = \frac{2\sigma - b}{\sqrt{\sigma}}.$$

Система уравнений {eq1_0,eq2_0} описывает колебания автоматически регулируемого маятника с энергией:

$$H = \frac{\dot{x}^2}{2} + \frac{x^4}{4} + \frac{(q-1)x^2}{2}.$$

Здесь q – регулирующий параметр. Кроме того, на маятник действует сила трения: $-\varepsilon h\dot{x}$. Уравнение eq2_0 описывает обратную связь регулятора с маятником.

Рассмотрим сначала случай отсутствия обратной связи ($\varepsilon=0$). Соответствующая система двух дифференциальных уравнений после замены $\dot{x}(t) = y(t)$ принимает вид (для подстановки используется команда **subs**):

```
> eq3:=diff(x(t),t)=y(t);
> eq1:=subs(epsilon=0,eq3,eq1_0);
```

$$eq3 := \frac{\partial}{\partial t}\,x(t) = y(t)$$

$$eq1 := \left(\frac{\partial}{\partial t}\,y(t)\right) + x(t)^3 + (q(t)-1)\,x(t) = 0$$

В этом случае управляющий параметр q постоянен и определяется только начальными данными. Потенциальная энергия маятника без обратной связи имеет вид:

```
> PEnerg:=x^4/4+(qq-1)*x^2/2;
```

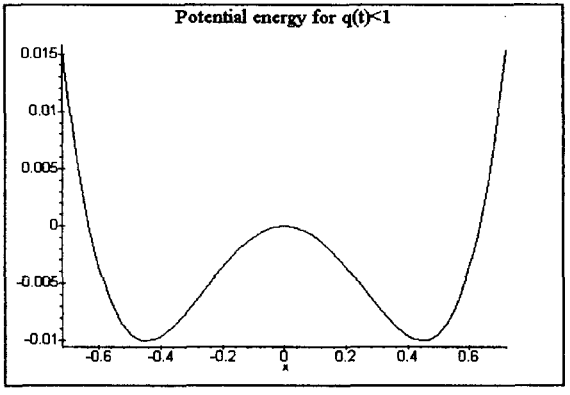
$$PEnerg := \frac{1}{4}x^4 + \frac{1}{2}(qq-1)x^2$$

Здесь для управляющего параметра введено новое обозначение *qq*.

При исследовании поведения маятника в отсутствие обратной связи достаточно рассмотреть три качественно различных случая: a) параметр $qq<1$, b) параметр $qq=1$, c) параметр $qq>1$.

Пусть $qq<1$; присвоим параметру *qq* значение 0.8 и воспользуемся командой **subs** для подстановки этого значения в уравнения. График потенциальной энергии в этом случае имеет вид:

```
> plot(subs(qq=0.8,PEnerg),
>      x=-0.72..0.72,axes=FRAMED,
>      title=`Potential energy for q(t)<1`);
```



Видно, что потенциальная энергия имеет два минимума, которым соответствуют устойчивые точки покоя (равновесия) и локальный максимум, которому отвечает неустойчивая точка покоя маятника (седловое равновесие).

Характерный фазовый портрет изображен при помощи команды из пакета **DEtools**.

```
> with(DEtools):
> phaseportrait([eval(-subs(y(t)=Y,rhs(eq3))),
>   eval(subs(y(t)=Y,q(t)=0.8,x(t)=X,lhs(eq1)))],
>   [X,Y],0..30,{[0,-0.65,0],[0,-0.3,0],
>   [0,-0.15,0.0],[0,-0.01,0.0],[0,0.65,0],
>   [0,0.3,0],[0,0.15,0.0],[0,0.01,0.0]},
```

```
> stepsize=0.25, title=
> `The phase portrait for q(t)<1`,axes=BOXED);
```

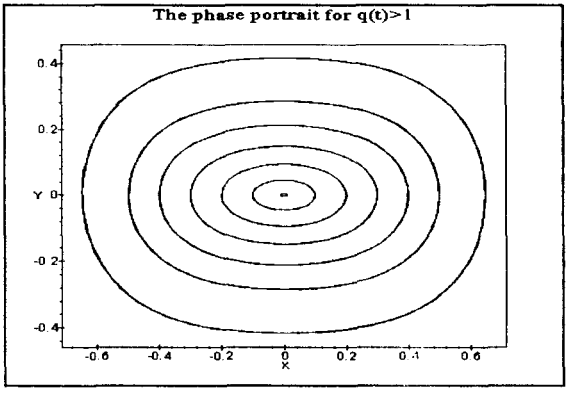
Фазовая плоскость разбита сепаратрисами седлового нулевого равновесия на три области с различным поведением траекторий: две области с замкнутыми траекториями, обращающимися вокруг соответствующего устойчивого равновесия, и внешняя по отношению к сепаратрисному контуру область.

Аналогично для случая с) при qq=1.2 строятся график потенциальной энергии и фазовый портрет системы.

```
> plot(subs(qq=1.2,PEnerg),
>      x=-0.72..0.72,axes=FRAMED,
>      title=`Potential energy for q(t)>1`);
```

Здесь глобальному минимуму потенциальной энергии отвечает единственное устойчивое нулевое равновесие.

```
> phaseportrait([eval(-subs(y(t)=Y,rhs(eq3))),
>   eval(subs(y(t)=Y,q(t)=1.2,x(t)=X,lhs(eq1)))],
>   [X,Y],0..30,{[0,-0.65,0],[0,-0.5,0],
>   [0,-0.4,0.0],[0,-0.3,0.0],[0,-0.2,0],
>   [0,-0.1,0],[0,0.01,0.0]},stepsize=0.25,
>   title=`The phase portrait for q(t)>1`,
>   axes=BOXED);
```



В этом случае единственное нулевое равновесие является центром, а все траектории замкнуты.

Данная система при ε=0 интегрируется явно. Для аналитического решения уравнения eq_1 воспользуемся командой **dsolve**. Для случая b) ($q(t)$=1) решение имеет вид:

```
> eq1:=subs(epsilon=0,q(t)=1,eq1_0);
> dsolve({eq1},{x(t)});
```

$$eq1 := \left(\frac{\partial^2}{\partial t^2}\,x(t)\right) + x(t)^3 = 0$$

$$t = \int_0^{x(t)} -2\,\frac{1}{\sqrt{-2\,y1^4 + 4\,\_C1}}\,dy1 - \_C2,\ t = \int_0^{x(t)} 2\,\frac{1}{\sqrt{-2\,y2^4 + 4\,\_C1}}\,dy2 - \_C2$$

Для случая a) при qq=0.8 получаем:

```
> eq1:=subs(epsilon=0,q(t)=8/10,eq1_0);
> dsolve({eq1},{x(t)});
```

$$eq1 := \left(\frac{\partial^2}{\partial t^2} x(t)\right) + x(t)^3 - \frac{1}{5} x(t) = 0$$

$$t = \int_0^{x(t)} 10 \frac{1}{\sqrt{-50\,y1^4 + 20\,y1^2 + 200\,\_C1}}\, dy1 - \_C2,$$

$$t = \int_0^{x(t)} -10 \frac{1}{\sqrt{-50\,y2^4 + 20\,y2^2 + 200\,\_C1}}\, dy2 - \_C2$$

Аналогично получается решение в случае c) при qq=1.2. Для этого достаточно двух команд (вывод результата опустим).

```
> eq1:=subs(epsilon=0,q(t)=12/10,eq1_0);
> dsolve({eq1},{x(t)});
```

Далее рассмотрим полную систему ($\varepsilon \neq 0$). Сделав замену $\dot{x}(t) = y(t)$, получим систему трех обыкновенных дифференциальных уравнений первого порядка:

```
> eq3:=diff(x(t),t)=y(t);
> eq1:=subs(eq3,eq1_0);
> eq2:=subs(eq3,eq2_0);
```

$$eq3 := \frac{\partial}{\partial t} x(t) = y(t)$$

$$eq1 := \left(\frac{\partial}{\partial t} y(t)\right) + \varepsilon\, h\, y(t) + x(t)^3 + (q(t) - 1)\, x(t) = 0$$

$$eq2 := \frac{\partial}{\partial t} q(t) = -\varepsilon\, a\, q(t) + \varepsilon\, \beta\, x(t)^2$$

Найдем равновесия системы уравнений и исследуем их устойчивость. Обозначим координаты точек равновесия через *equil_X*, *equil_Y*, *equil_Q*. Уравнения для определения равновесий имеют вид:

```
> eqr1:=eval(subs(x(t)=X,y(t)=Y,q(t)=Q,
>                 rhs(eq3)-lhs(eq3)));
> eqr2:=eval(subs(x(t)=X,y(t)=Y,q(t)=Q,
>                 rhs(eq1)-lhs(eq1)));
> eqr3:=eval(subs(x(t)=X,y(t)=Y,q(t)=Q,
```

$$eqr1 := Y$$

$$eqr2 := -\varepsilon\, h\, Y - X^3 - (Q-1)X$$

$$eqr3 := -\varepsilon\, a\, Q + \varepsilon\, \beta\, X^2$$

Из уравнений *eqr1* и *eqr3* находим:

```
> equil_Y:=solve({eqr1},Y);
```

$$equil\_Y := \{Y = 0\}$$

```
> equil_Q:=solve({eqr3},Q);
```

$$equil\_Q := \left\{Q = \frac{\beta X^2}{a}\right\}$$

После подстановки полученных выражений в уравнение *eqr2* получим:

```
> equil_X:=solve({subs(equil_Y,equil_Q,eqr2)},X);
```

$$equil\_X := \{X = 0\}, \left\{X = \frac{\sqrt{\frac{a+\beta}{a}}}{-1-\frac{\beta}{a}}\right\}, \left\{X = -\frac{\sqrt{\frac{a+\beta}{a}}}{-1-\frac{\beta}{a}}\right\}$$

У системы всегда существует нулевое равновесие, а анализируя координаты ненулевых равновесий, несложно выяснить области их существования. Всего у системы уравнений (2) может быть три равновесия. Занесем их координаты в массив *equil*:

```
> equil:=array(1..3):
> for i from 1 to 3 do equil[i]:={} union
>     equil_X[i] union subs(equil_X[i],equil_Q)
>     union equil_Y; od;
```

$$equil_1 := \{Y = 0, X = 0, Q = 0\}$$

$$equil_2 := \left\{Y = 0, X = \frac{\sqrt{\frac{a+\beta}{a}}}{-1-\frac{\beta}{a}}, Q = \frac{\beta(a+\beta)}{\left(-1-\frac{\beta}{a}\right)^2 a^2}\right\}$$

$$equil_3 := \left\{Y = 0, X = -\frac{\sqrt{\frac{a+\beta}{a}}}{-1-\frac{\beta}{a}}, Q = \frac{\beta(a+\beta)}{\left(-1-\frac{\beta}{a}\right)^2 a^2}\right\}$$

Вычислим матрицу Якоби для системы уравнений при помощи команды из пакета `linalg`:

```
> with(linalg,jacobian):
> J:=jacobian([eqr1,eqr2,eqr3],[X,Y,Q]);
```

$$J := \begin{bmatrix} 0 & 1 & 0 \\ -3X^2 - Q + 1 & -\varepsilon h & -X \\ 2\varepsilon\beta X & 0 & -\varepsilon a \end{bmatrix}$$

Для исследования устойчивости равновесий напишем процедуру со следующими входными параметрами: *ve* – значение параметра ε, *va* – значение параметра *a*, *vbeta* – значение параметра *β*, *vh* – значение параметра *h* и *equilibr* – координаты точки равновесия. Процедура вычисляет характеристический многочлен матрицы линеаризации в окрестности равновесия, находит его корни, сортирует их в порядке возрастания и анализирует устойчивость равновесия. В результате работы процедуры выдается структура, первым элементом которой является булевская величина, равная `true`, если равновесие устойчиво, и `false`, если равновесие неустойчиво. Следующим элементом структуры является массив действительных частей корней характеристического полинома.

```
> eig:=proc(ve,va,vbeta,vh,equilibr)
>       local j,bo,eigv,chp,lambda;
>       chp:=factor(subs(equilibr,
>                linalg[charpoly](J,lambda)));
>       chp:=simplify(subs(epsilon=ve,
>                  a=va,beta=vbeta,h=vh,chp));
>       eigv:=evalf(solve(chp,lambda));
> j:=max(Re(eigv[1]),Re(eigv[2]),Re(eigv[3]));
> bo:=true; if evalf(j)>0 then bo:=false; fi;
> [bo,sort([eigv])];
> end;
```

```
eig :=
proc(ve,va,vbeta,vh,equilibr)
local j,bo,eigv,chp,lambda;
  chp := factor(subs(equilibr,linalg[charpoly](J,lambda)));
   chp := simplify(
     subs(epsilon = ve,a = va,beta = vbeta,h = vh,chp));
   eigv := evalf(solve(chp,lambda));
   j := max(Re(eigv[1]),Re(eigv[2]),Re(eigv[3]));
   bo := true;
   if 0 < evalf(j) then bo := false fi;
```

```
    [bo,sort([eigv])]
  end
```

Например, для значений параметров ε=1, a=0.1, β=0.1 и h=1 нулевое равновесие неустойчиво, а ненулевое равновесие *equil*[2] устойчиво.

```
> subs(epsilon=1,a=1/10,beta=1/10,h=1,equil[2]);
```

$$\left\{Y = 0, X = -\frac{1}{2}\sqrt{2}, Q = \frac{1}{2}\right\}$$

```
> eig(1,0.1,0.1,1,equil[1]);
> eig(1,0.1,0.1,1,equil[2]);
```

```
                [false, [-1.618033989, -.1000000000, .6180339890]]
[true, [-.4396109657 + .8441737260 I, -.4396109657 - .8441737260 I, -.2207780688]]
```

Теперь присвоим параметрам те значения, при которых система Лоренца демонстрирует хаотическое поведение.

```
> b:=8/3: r:=28: sigma:=10:
> epsilon:=1/sqrt(r-1); h:=(sigma+1)/sqrt(sigma);
> a:=b/sqrt(sigma);beta:=(2*sigma-b)/sqrt(sigma);
```

$$\varepsilon := \frac{1}{9}\sqrt{3}$$

$$h := \frac{11}{10}\sqrt{10}$$

$$a := \frac{4}{15}\sqrt{10}$$

$$\beta := \frac{26}{15}\sqrt{10}$$

Для численного решения задачи Коши при заданных значениях параметров используем команду **dsolve** с параметром **numeric**.

```
> nsol:=dsolve({eq1,eq2,eq3,x(0)=-3/10,
>         y(0)=-1/10, q(0)=1/2},{x(t),y(t),q(t)},
>         type=numeric, method=rkf45,
>         relerr=0.00001,abserr=0.000001,
>         output=listprocedure);
```

nsol := [*t* = *proc*(*t*) ... *end*, *x*(*t*) = *proc*(*t*) ... *end*,
y(*t*) = *proc*(*t*) ... *end*, *q*(*t*) = *proc*(*t*) ... *end*]

Обратим внимание на то, что форма вывода задается опцией **output**. При помощи команды **odeplot** из пакета **plots** построим

изображение полученной хаотической траектории в трехмерном пространстве:

```
> with(plots,odeplot):
> odeplot(nsol, [x(t),y(t),q(t)],0..250,
>         numpoints=800,color=x(t),
>         axes=FRAME,labels=[x,y,q],
>         style=line, orientation=[100,60],title=
>         `The strange attractor`,color=black);
```

Далее перейдем к анализу поведения решения во времени и изучению динамики потенциальной энергии, которая для системы с обратной связью зависит от изменяющегося управляющего параметра. Для численного решения системы уравнений используем специальный формат команды **dsolve**, позволяющий получать решение в виде таблицы. Для этого нужно указать параметр **value=tm**, где в массиве *tm* содержатся значения моментов времени, в которые будут рассчитаны переменные *x*,*y* и *q*. В результате работы команды **dsolve** в переменной **nsola** будет находиться таблица, содержащая значения переменных в указанных точках.

Обратимся к команде **dsolve** и напечатаем первые пять вычисленных точек траектории.

```
> tm:=array(1..301):
> for i from 1 to 301 do tm[i]:=(i-1)/300*150;od:
> nsola:=dsolve({eq1,eq2,eq3),
>                x(0)=-3/10,y(0)=-1/10,q(0)=1/2},
>                {x(t),y(t),q(t)},
```

```
>                   type=numeric, method=rkf45,
>                   relerr=0.00001,
>                   abserr=0.000001,value=tm):
> print(nsola[1,1]);
> for i from 1 to 5 do
>     print(nsola[2,1][i,j] $j=1..4); od:
                    [t  y(t)  q(t)  x(t)]
   0, -.1000000000000000, .5000000000000000, -.3000000000000000
 .5000000000000000, -.1251699174599914, .5154682461277501, -.3563880635830255
  1., -.1414903722502397, .5524387359616295, -.4235659228485054
 1.500000000000000, -.1406767970710631, .6167084766522043, -.4950376243845029
  2., -.1125780876352003, .7105651144963369, -.5596712499094820
```

Сформируем массивы, содержащие значения переменных x (массив *xm*), y (массив *ym*), q (массив *qm*) и потенциальной энергии (массив *pem*) в указанных точках (массив *tm*).

```
> xm:=array(1..301): ym:=array(1..301):
> qm:=array(1..301): pem:=array(1..301):
> for i from 1 to 301 do
>   xm[i]:=nsol1[2,1][i,2];
>   qm[i]:=nsol1[2,1][i,3];
>   ym[i]:=nsol1[2,1][i,4];
>   pem[i]:=evalf(subs(qq=qm[i],x=xm[i],PEnerg));
> od:
```

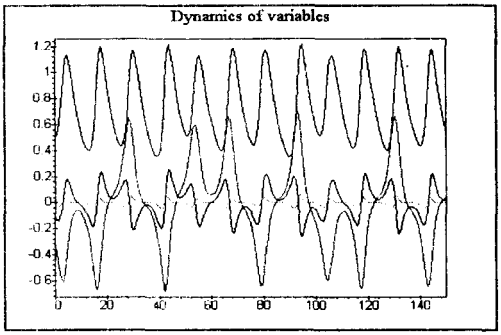
Теперь приблизим полученное численное решение линейными сплайнами, для чего вызовем соответствующую команду **spline**. Определенные в результате действия команды **spline** процедуры *spx*, *spy*, *spq* и *sppe* позволяют получать приближенные значения переменных x, y, q и потенциальной энергии при любых значениях переменной t на заданном отрезке.

```
> readlib(spline):
> spline(tm,xm,t,linear):
> spx:=`spline/makeproc`(",t):
> spline(tm,ym,t,linear):
> spy := `spline/makeproc`(",t):
> spline(tm,qm,t,linear):
> spq := `spline/makeproc`(",t):
> spline(tm,pem,t,linear):
```

```
> sppe := `spline/makeproc`(",t):
```

График зависимости переменных от времени на интервале $t \in [0,150]$ выглядит следующим образом:

```
> plot({spx,spy,spq,sppe},0..150,axes=BOXED,
>       title=`Dynamics of variables`);
```



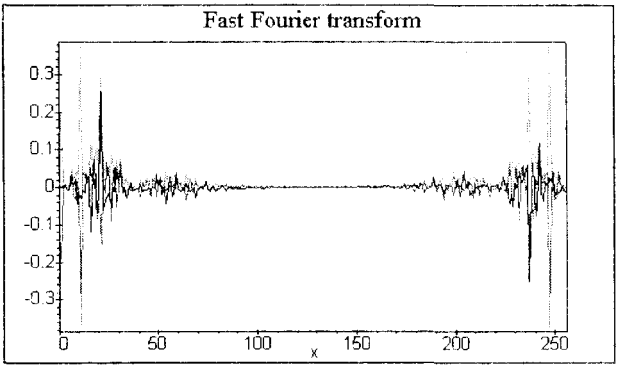
Одним из критериев хаотичности решения системы дифференциальных уравнений является сложность его фурье-спектра. Применим быстрое преобразование Фурье (команда **FFT**) для обработки полученного временного ряда значений потенциальной энергии по 256 точкам.

```
> xvar:=array(1..256): yvar:=array(1..256):
> mvar:=array(1..256):
> for i from 1 to 256 do
>        xvar[i]:=pem[i]; yvar[i]:=0; od:
> readlib(FFT):
> FFT(8,xvar,yvar);
                              256
> for i from 1 to 256 do
>        mvar[i]:=sqrt(xvar[i]^2+yvar[i]^2); od:
```

Построим в одних осях координат графики действительной и мнимой части фурье-спектра, а также их модуля. Как видно из приводимого рисунка, полученное решение действительно является хаотическим.

```
> plot({[[k,xvar[k]] $k=1..256],
>       [[l,yvar[l]] $l=1..256],
```

```
>        [[m,mvar[m]] $m=1..256]},
>        x=0..256,axes=BOXED,
>        title=`Fast Fourier transform`);
```



Для данной задачи имеется интересная физическая интерпретация. Эта система дифференциальных уравнений описывает движение идеального шарика (точки) в меняющейся во времени потенциальной яме. Напомним, что форма самой потенциальной ямы зависит от управляющего параметра q. Maple позволяет представить решение в удобном для физической трактовки виде.

Изобразим на одном рисунке меняющуюся во времени потенциальную яму и траекторию движения шарика. Вначале определим процедуру-функцию, которая выдает значение потенциальной энергии в точке с координатой x в момент времени t.

```
> poten:=(x,t)->x^4/4+(spq(t)-1)*x^2/2;
```

$$poten := (x,t) \to \frac{1}{4}x^4 + \frac{1}{2}(\mathrm{spq}(t)-1)\,x^2$$

Переменной *pic1* присвоим графический образ поверхности, которая определяется значениями потенциальной энергии на временном отрезке $t \in [0,150]$ и интервале координаты $x \in [-0.8,0.8]$.

```
> pic1:=plot3d(poten,-0.8..0.8,0..150,
>              style=WIREFRAME,numpoints=3000):
```

Переменной *pic2* присвоим изображение траектории материальной точки (шарика), движущейся в потенциальной яме. Здесь координаты траектории определяются переменными x, t и потенциальной энергией.

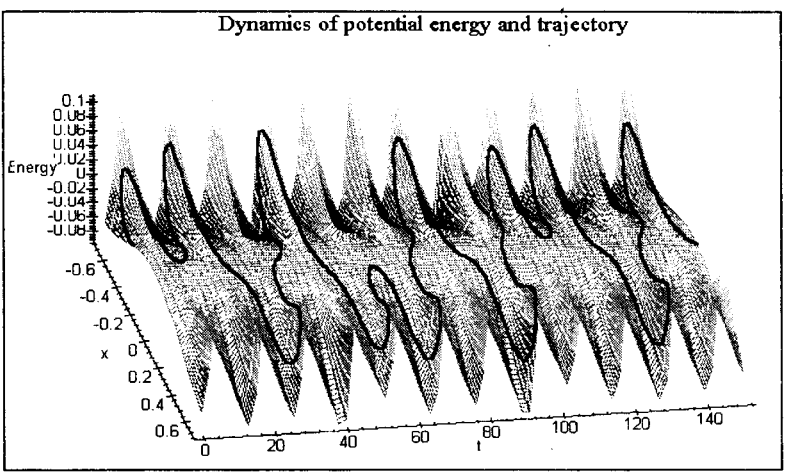
```
> with(plots,spacecurve):
> timet:=t->t;
```

$$timet := t \to t$$

```
> pic2:=spacecurve([spx,timet,sppe],0..150,
>                  numpoints=400,
>                  color=black,thickness=2):
```

При помощи команды **display3d** получим рисунок, на котором изображены форма потенциальной ямы при изменении времени и траектория шарика.

```
> with(plots,display3d):
> display3d({pic1,pic2},orientation=[-18,35],
>         title=`Dynamic of potengial
>         energy and trajectory`,
>         axes=FRAMED,labels=[x,t,Energy]);
```



Завершим это мини-исследование иллюстрацией анимационных возможностей Maple. Следующие команды позволяют подготовить мультипликацию движения шарика в изменяющейся потенциальной яме. Для этого сначала опишем процедуру для изображения шарика в виде окружности с центром в точке (x,y)

```
> ball := proc(x,y) local i;
>     PLOT(POLYGONS(
>     [seq([ evalf(x+0.05*cos(Pi*i/10)/2),
>     evalf(0.006+y+0.01*sin(Pi*i/10)/2) ],
```

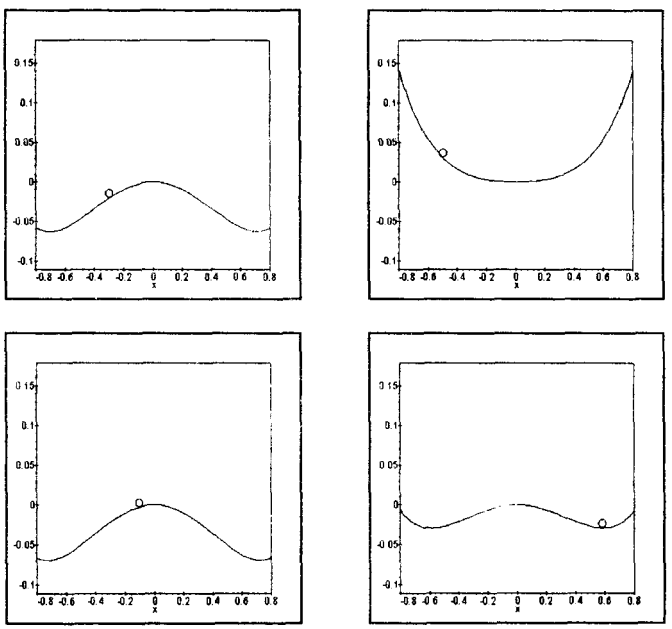
```
>      i = 1..20)]));
>  end:
```

Затем подготовим процедуру для рисования шарика и графика потенциальной энергии *PEnerg*. Параметром процедуры является номер вычисленной точки решения, а в теле процедуры используются отвечающие *n* значения переменной *x* и потенциальной энергии – *xm*[*n*] и *pem*[*n*] соответственно.

```
> with(plots,display):
> figpic:=proc(n:integer);
>       display({plot(subs(qq=qm[n],PEnerg),
>               x=-0.8..0.8),ball(xm[n],pem[n])});
> end:
```

Для запуска подготовленного мультфильма, изображающего движения шарика в потенциальной яме, теперь достаточно запустить команду **display** с параметром **insequence=true**. На рисунке представлено несколько кадров этого фильма.

```
> display([seq(figpic(i),i=1..150)],
>           insequence=true,axes=BOXED);
```



10.4. Течение вязкой жидкости в канале

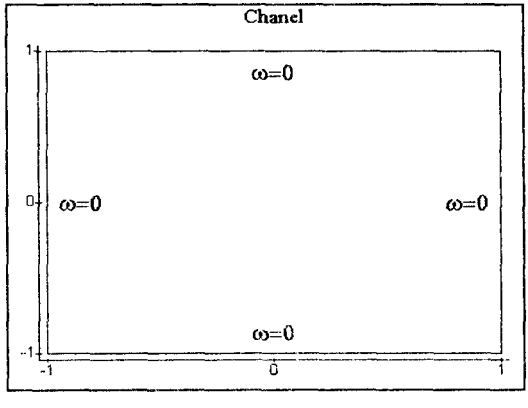
В данном разделе дано применение системы Maple для решения уравнений в частных производных методом Галеркина. Рассмотрена модельная задача о стационарном течении вязкой жидкости в канале с квадратным поперечным сечением. Эта задача часто используется в качестве тестовой для проверки численных методов и пакетов [9].

Уравнение для компоненты скорости, направленной вдоль канала, в безразмерных координатах имеет вид:

```
> eq:=diff(omega(x,y),x$2)+diff(omega(x,y),y$2)+1;
```

$$eq := \left(\frac{\partial^2}{\partial x^2}\omega(x,y)\right)+\left(\frac{\partial^2}{\partial y^2}\omega(x,y)\right)+1$$

На границе $x=\pm1$, $y=\pm1$ скорость обращается в нуль ($\omega=0$). С использованием графических структур и команд изобразим схематически сечение канала и граничные условия (текст команд опускаем).



Решение уравнения будем разыскивать в виде ряда по линейно-независимым функциям, удовлетворяющим граничным условиям. Для обозначения этих функций используем имя *basf*(*i,j,x,y*), где (*i,j*) – определяет номер базисной функции, а *x,y* – координаты.

```
> Omega:=Sum(Sum(a[i,j]*basf(i,j,x,y),`i`=1..N),
                 `j`=1..M);
```

$$\Omega := \sum_{j=1}^{M} \left(\sum_{i=1}^{N} a_{i,j} \, \mathrm{basf}(i, j, x, y) \right)$$

Здесь $a_{i,j}$ – неизвестные коэффициенты разложения по функциям *basf*. Вначале в качестве базисных функций выберем тригонометрические функции вида:

```
> basf:=proc(n,m,x,y);
>    cos((2*n-1)*Pi/2*x)*cos((2*m-1)*Pi/2*y); end;
```

basf := proc(n,m,x,y)
cos(1/2(2*n-1)*Pi*x)*cos(1/2*(2*m-1)*Pi*y)*
end

Убедимся, что эти функции действительно удовлетворяют граничным условиям и являются линейно-независимыми:

```
> evalf(simplify(basf(1,2,1,-1)));
                              0
> evalf(simplify(basf(1,2,-1,1)));
                              0
> int(int(basf(1,1,x,y)*basf(3,2,x,y),x=-1..1),
>                          y=-1..1);
                              0
```

Ограничимся четырьмя членами ряда; тогда приближенное решение *omega_n* выглядит следующим образом:

```
> n:=2: m:=2:
> omega_n:=subs(N=n,M=m,value(Omega));
```

$$omega\_n := \sum_{j=1}^{2} \left(\sum_{i=1}^{2} a_{i,j} \cos\left(\frac{1}{2}(2i-1)\pi x\right) \cos\left(\frac{1}{2}(2j-1)\pi y\right) \right)$$

Подставив приближенное решение в исходное уравнение, получим невязку:

```
> eq_a:=subs(omega(x,y)=omega_n,eq);
```

$$eq\_a := \left(\frac{\partial^2}{\partial x^2}\left(a_{1,1}\cos\left(\frac{1}{2}\pi x\right)\cos\left(\frac{1}{2}\pi y\right)\right.\right.$$
$$+a_{2,1}\cos\left(\frac{3}{2}\pi x\right)\cos\left(\frac{1}{2}\pi y\right)$$
$$+a_{1,2}\cos\left(\frac{1}{2}\pi x\right)\cos\left(\frac{3}{2}\pi y\right)$$
$$\left.\left.+a_{2,2}\cos\left(\frac{3}{2}\pi x\right)\cos\left(\frac{3}{2}\pi y\right)\right)\right)+\left(\frac{\partial^2}{\partial y^2}\right($$
$$a_{1,1}\cos\left(\frac{1}{2}\pi x\right)\cos\left(\frac{1}{2}\pi y\right)+a_{2,1}\cos\left(\frac{3}{2}\pi x\right)\cos\left(\frac{1}{2}\pi y\right)$$
$$+a_{1,2}\cos\left(\frac{1}{2}\pi x\right)\cos\left(\frac{3}{2}\pi y\right)$$
$$\left.\left.+a_{2,2}\cos\left(\frac{3}{2}\pi x\right)\cos\left(\frac{3}{2}\pi y\right)\right)\right)+1$$

В качестве поверочных функций выберем те же функции *basf*, что используются в приближенном решении. Для нахождения неизвестных коэффициентов $a_{i,j}$ нужно решить систему линейных алгебраических уравнений, получаемую после проекции невязки на поверочные функции. Удобно организовать эти вычисления в виде процедуры, входными параметрами которой являются невязка *eq_a*, целые числа *n* и *m*, выражение для приближенного решения *omega_n* и булевский параметр *bol*. При значении параметра *bol*=**true** выдаются на экран промежуточные результаты, а при *bol*=**false** – нет.

Поясним работу процедуры. Система линейных алгебраических уравнений формируется в результате проектирования на поверочные функции; здесь для вычисления скалярного произведения используется команда **int**. Затем решение этой системы находится при помощи команды **solve** и подставляется в выражение для приближенного решения.

```
> sol_gal:=proc(eq_a,n,m,omega_n,bol:boolean)
> local eq_coef,var_coef,solc_a,i,j;
> eq_coef:={}; var_coef:={};
> for i from 1 to n do
> for j from 1 to m do
>     eq_coef:=eq_coef union
>              {int(int(eval(eq_a)*basf(i,j,x,y),
>               x=-1..1),y=-1..1)};
```

```
>     var_coef:=var_coef union {a[i,j]};
> od; od;
> if bol then print(eval(eq_coef));
>             print(eval(var_coef)); fi;
> solc_a:=solve(eq_coef,var_coef);
> if bol then print(solc_a); fi;
> subs(solc_a,omega_n); end:
```

После обращения к данной процедуре получим искомое приближенное решение :

```
> omega_a:=sol_gal(eq_a,n,m,omega_n,true);
```

$$\left\{-\frac{1}{6}\frac{32+15a_{1,2}\pi^4}{\pi^2},-\frac{1}{2}\frac{a_{1,1}\pi^4-32}{\pi^2},-\frac{1}{6}\frac{15a_{2,1}\pi^4+32}{\pi^2},-\frac{1}{18}\frac{-32+81a_{2,2}\pi^4}{\pi^2}\right\}$$

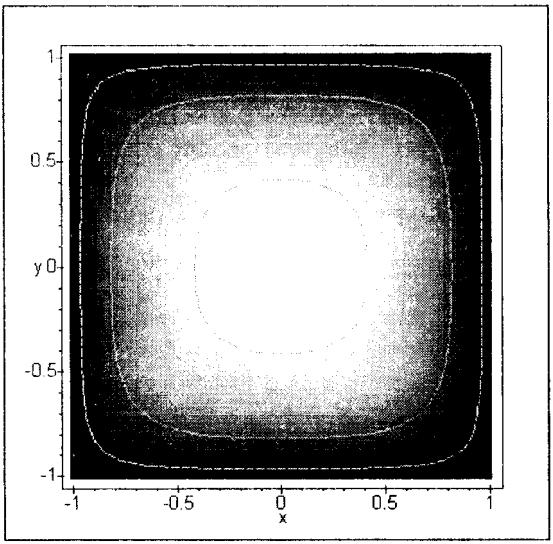
$$\left\{a_{1,1},a_{2,1},a_{1,2},a_{2,2}\right\}$$

$$\left\{a_{2,2}=\frac{32}{81}\frac{1}{\pi^4},a_{1,2}=-\frac{32}{15}\frac{1}{\pi^4},a_{2,1}=-\frac{32}{15}\frac{1}{\pi^4},a_{1,1}=32\frac{1}{\pi^4}\right\}$$

$$omega\_a := 32\frac{\cos\left(\frac{1}{2}\pi x\right)\cos\left(\frac{1}{2}\pi y\right)}{\pi^4}-\frac{32}{15}\frac{\cos\left(\frac{3}{2}\pi x\right)\cos\left(\frac{1}{2}\pi y\right)}{\pi^4}-\frac{32}{15}\frac{\cos\left(\frac{1}{2}\pi x\right)\cos\left(\frac{3}{2}\pi y\right)}{\pi^4}+\frac{32}{81}\frac{\cos\left(\frac{3}{2}\pi x\right)\cos\left(\frac{3}{2}\pi y\right)}{\pi^4}$$

Изобразим полученное решение графически при помощи команд пакета `plots`. На рисунке изображены плотность функции скорости и несколько изолиний.

```
> pic1:=plots[densityplot](omega_a,
>             x=-1..1,y=-1..1,grid=[60,60],
>             style=PATCHNOGRID,axes=BOXED):
> pic2:=plots[implicitplot]({omega_a=0.02,
>          omega_a=0.1,omega_a=0.2,omega_a=0.25},
>          x=-1..1,y=-1..1,
>          grid=[40,40],color=khaki):
> plots[display]({pic1,pic2},axes=BOXED);
```



Точное решение для скорости ω в точке с координатами (0,0) и для безразмерного расхода q возьмем из книги [9]: $\omega(0,0) = 0.2947$, $q = 0.5623$. Расход q определяется как интеграл от функции скорости ω по области сечения. Сравним полученное приближенное значение с точным.

```
> evalf(subs(x=0,y=0,omega_a));
                              .2887656041
```

```
> Int(Int(omega(x,y),x=-1..1),y=-1..1)=
> evalf(int(int(omega_a,x=-1..1),y=-1..1));
```

$$\int_{-1}^{1}\int_{-1}^{1} \omega(x,y)\,dx\,dy = .5569626645$$

Видно, что всего четыре члена в галеркинском разложении по тригонометрическим функциям дают неплохой результат.

Решения могут быть получены и для другой системы линейно-независимых функций. Например, используем для этого полиномы вида:

```
> basf:=proc(n,m,x,y);
>                (1-x^2)^n*(1-y^2)^m; end:
```

Приближенное решение в этом случае имеет вид:

```
> omega_n:=subs(N=n,M=m,value(Omega));
```

$$omega\_n := \sum_{j=1}^{2}\left(\sum_{i=1}^{2} a_{i,j}\left(1-x^2\right)^i\left(1-y^2\right)^j\right)$$

После подстановки приближенного решения в уравнение, находим невязку:

```
> eq_b:=expand(subs(omega(x,y)=omega_n,eq));
```

$$\begin{aligned} eq\_b := {} & 1 + 12a_{2,2}x^4y^2 + 12a_{2,2}x^2y^4 - 48a_{2,2}x^2y^2 - 12a_{1,2}x^2y^2 \\ & - 12a_{2,1}x^2y^2 - 4a_{1,1} - 6a_{2,1} - 6a_{1,2} - 8a_{2,2} - 2a_{1,2}y^4 \\ & + 16a_{1,2}y^2 - 2a_{2,1}x^4 + 4a_{2,1}y^2 + 16a_{2,1}x^2 + 2a_{1,1}y^2 \\ & + 2a_{1,1}x^2 + 20a_{2,2}x^2 - 4a_{2,2}x^4 - 4a_{2,2}y^4 + 20a_{2,2}y^2 \\ & + 4a_{1,2}x^2 \end{aligned}$$

Обратившись к описанной выше процедуре *sol_gal*, получим приближенное решение по полиномам.

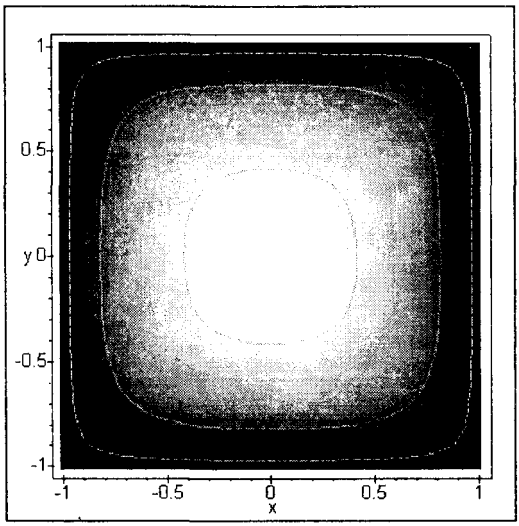
```
> omega_b:=sol_gal(eq_b,n,m,omega_n,false);
```

$$\begin{aligned} omega\_b := {} & \frac{1}{2}\left(1-x^2\right)\left(1-y^2\right) - \frac{21}{128}\left(1-x^2\right)^2\left(1-y^2\right) \\ & - \frac{21}{128}\left(1-x^2\right)\left(1-y^2\right)^2 + \frac{63}{512}\left(1-x^2\right)^2\left(1-y^2\right)^2 \end{aligned}$$

График плотности функции скорости *omega_b* и некоторые ее изолинии приведены на следующем рисунке:

```
> pic1:=plots[densityplot](omega_b,
>              x=-1..1,y=-1..1,grid=[60,60],
>              style=PATCHNOGRID,axes=BOXED):
> pic2:=plots[implicitplot]({omega_b=0.02,
>          omega_a=0.1,omega_a=0.2,omega_a=0.25},
```

```
>          x=-1..1,y=-1..1,
>          grid=[40,40],color=khaki):
> plots[display]({pic1,pic2},axes=BOXED);
```



Следующие две команды позволяют сравнить полученное полиномиальное разложение с точным решением.

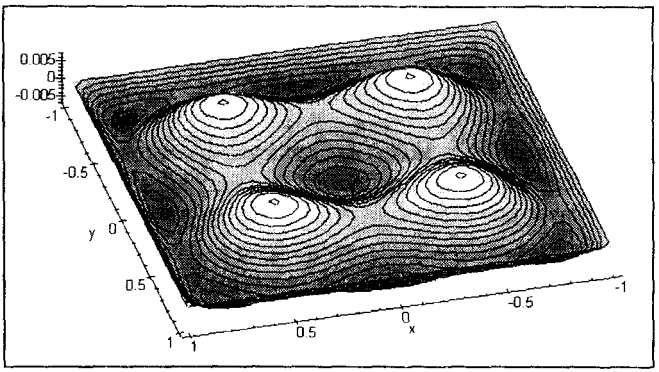
```
> evalf(subs(x=0,y=0,omega_b));
                          .2949218750
```

```
> Int(Int(omega(x,y),x=-1..1),y=-1..1)=
> evalf(int(int(omega_b,x=-1..1),y=-1..1));
```

$$\int_{-1}^{1}\int_{-1}^{1} \omega(x,y)\,dx\,dy = .5622222222$$

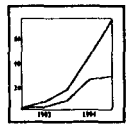
Видно, что при одинаковом числе членов ряда разложение по полиномам дает лучший результат. Разность между двумя приближенными решениями *omega_a* и *omega_b* приведена на следующем рисунке.

```
> plot3d(omega_a-omega_b,x=-1..1,y=-1..1,
>        axes=FRAMED,orientation=[75,12],
>        style=PATCHCONTOUR,grid=[40,40]);
```



Увеличивая число членов ряда в полиномиальном галеркинском разложении, убедимся, что уже для 25 членов получается практически точное решение.

```
> n:=5: m:=5:
> omega_n:=subs(N=n,M=m,value(Omega)):
> eq_c:=subs(omega(x,y)=omega_n,eq):
> omega_c:=sol_gal(eq_c,n,m,omega_n,false):
> evalf(subs(x=0,y=0,omega_c)):
                              .2946866894
> evalf(int(int(omega_c,x=-1..1),y=-1..1));
                              .5623078843
```



11. Заключение

Данная книга по замыслу авторов должна помочь русскоязычным математикам в освоении замечательной системы аналитических вычислений – удобного, мощного и интеллектуального пакета Maple , действительно облегчающего жизнь.

Написанное не есть руководство пользователя, а является, скорее всего, кратким справочником, отчасти обзором, по существу, сборником примеров и, возможно, брошюрой с рецептами. Авторам представляется, что для знакомства с пакетом, для введения в его возможности и среду, это правильный подход, хотя они и понимают, что люди разнообразны – и одни утомятся от подробности примеров, другие возмутятся неполнотой описания спецификаций, третьи... Известны два мудрых высказывания Мао Цзе Дуна: "Читай, не читай – императором не станешь" и "Чтобы научиться плавать – надо плавать". Начните применять Maple и через некоторое время придут опыт и интуиция, столь необходимые в общении с такой развитой системой, как Maple.

Внимательный читатель, дочитавший до этого места, наверное, обратил внимание, что авторам близка тематика исследований динамических систем и интересно применение Maple в обучении. Это сказалось на выборе примеров мини-исследований и на отборе материала для книги. Желая помочь читателю, не обнаружившему "своих" глав, мы приводим в данном заключении некоторые Internet-адреса – своеобразный кончик нити, потянув за который, можно добрать необходимые сведения: ссылки на книги, архивы, программы. Кроме того, в Заключении помещена информация о сопутствующих пакету Maple околоматематических вещах: новостях, WWW, периодике и др.

11.1. Maple на марше

Как представляется авторам, пакету Maple гарантирована долгая и разнообразная жизнь. Во-первых, об этом свидетельствует частота реализаций версии V: первая состоялась в 1990 г., третья – в 1994, четвертая – в 1995. А там появится версия VI и т.д. Во-вторых, примечательно включение символьного анализатора Maple в состав инженерного пакета MathCad (компания MathSoft Inc.) и пакета

вычислительной математики MatLab (компания MathWorks). Далее, текстовый редактор Scientific Word фирмы TCI Software Research в результате скрещения с подвидом пакета Maple превратился в Scientific WorkPlace - рабочее место исследователя, где совмещены операции написания формул, проведения выкладок и численных расчетов, графического оформления результатов и подготовки статьи в формате LATEXa. Идея единой рабочей среды, т.е. дооснащения замечательного математического пакета современными средствами редактирования, близка авторам Maple, так что реализация ее не за горами.

Гексли говорил, что математика подобна жерновам: засыпаешь зерно – получаешь муку, а засыпаешь камни – получаешь песок. Возможности Maple перемалывать проблемы также растут. Любителям текстовых процессоров фирмы Microsoft, по-видимому, будет приятно узнать, что появился Math Office for Word 6.0 – инструмент, объединивший математический интеллект Maple с трудолюбием популярного редактора. Технология работы замечательно проста: не покидая созидаемого текста можно обращаться за скорой математической помощью Maple. В меню Word 6.0 после справки (Help) появился пункт Maple, апеллируя к которому или нажимая горячие клавиши (чистые руки и свежая голова настоятельно рекомендуются), вы отправляете свои выражения DDE-серверу Maple. Результаты аналитических преобразований, вычислений или графики выдаются прямо в редактируемый документ; с каждым документом связана своя сессия Maple.

Требования обычные: установленные Word 6.0 и Maple V Release 3 for Windows, 8 Мегабайт оперативной памяти, процессор от 80386 и дополнительные 3 Мегабайта на вашем жестком диске.

Из последних новостей следует выделить объявленное в январе 1996 г. учреждение партнерства между компаниями CRC Press Inc. и Waterloo Maple Inc. с намерением произвести переворот в доступе к научной и технической информации. Предполагается соединение запасенного в справочниках CRC Press Inc. материала с алгоритмическими и визуализационными возможностями Waterloo Maple Inc.

Вначале последует серия компакт-дисков с возможностью интерактивного использования их содержимого (формул, уравнений, таблиц). Первый компакт реализует известный справочник Стандартные Математические Таблицы и Формулы. Затем выйдут справочники по химии и физике, электротехнике и др.

Объявлено также, что эти интерактивные издания будут использовать специализированные программные средства, в частности такие, как MathDoc, TableDoc, GraphDoc и ChemDoc.

11.2. Maple и университеты

Пакет ныне широко используется для преподавания математики во многих учебных заведениях мира. Фирма Waterloo Maple Inc., кроме студенческой версии Maple, существенно более дешевой в силу сокращения ряда умений, предлагает также специальные программы, поощряющие приобретение кампус-лицензии на университет или массовую закупку продукта по сниженной цене за штуку - соответственно Ньютон-план и Гаусс-план.

11.3. Maple и периодика

Во многих научных журналах можно встретить ссылки на помощь Maple при проведении выкладок. Имеются специальный журнал "Maple Technical Newsletter", публикующий статьи о возможностях Maple и различных приложениях пакета, газета "Dr. Roots", обеспечивающая текущее информирование пользователей и поклонников.

11.4. Maple и Internet

Разумеется, у фирмы Maple Software есть WWW-сервер, домашняя страница (HomePage) и разветвленное гипертекстовое наполнение. Обратившись к странице

```
http://www.maplesoft.com
```

вы получите доступ к разнообразной информации о фирме, о ее продуктах (не Maple'ом единым), о всевозможных событиях и пр. Особо отметим, что тут же даны сведения о Maple V Share Library – общедоступной библиотеке написанных пользователями документов (worksheet), процедур для различных приложений и статей. Здесь можно найти литературу по Maple (руководства и вводные тексты), а также примеры приложений для биологии и инженерных наук, пакеты по линейной алгебре, физике, обобщенным функциям и т.д.

Дадим несколько ссылок на адреса ftp-серверов, которые содержат эту информацию и ссылки на другие источники.

Адрес	TCP/IP	Страна	Каталог
ftp.maplesoft.com	192.139.233.5	Канада	pub/maple
daisy.uwaterloo.ca	129.97.140.58	Канада	maple
ftp.inria.fr	128.93.2.54	Франция	lang/maple
ftp.can.nl	192.16.187.3	Голландия	pub/maple-ftplib
ftp.inf.ethz.ch	129.132.40.35	Швейцария	pub/maple
ftp.ask.uni-karlsruhe.de		Германия	pub/maple/share
unix.hensa.ac.uk		Англия	mirrors/maple

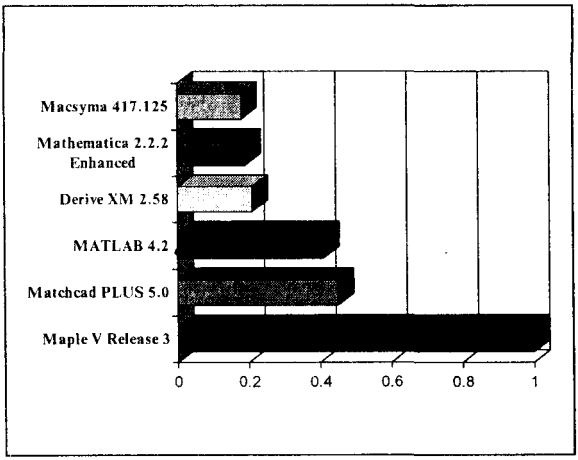
И еще один адрес богатого содержанием сервера:

```
http://www-users.informatik.rwth-aachen.de/~afw/maplev.html
```

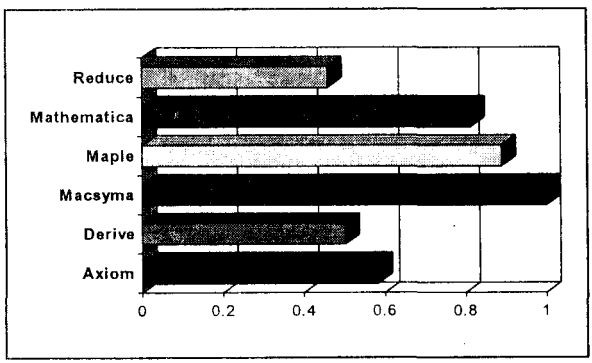
11.5. Maple и другие пакеты

Конечно, Maple не единственный пакет аналитических вычислений. Достаточно мощными конкурентами являются Macsyma, переживающая Ренессанс и решающая 95% примеров из справочника Камке, и замечательная Mathematica фирмы Wolfram Research. А кроме того, есть английский Axiom, удивительно небольшой и способный пакет Derive и др. Потенциальному пользователю такое соревнование систем, приводящее к развитию возможностей и умеренной ценовой политике, только на руку. Замечательные сравнительные обзоры пакетов даны в статьях [10,11]. Изучение этих статей и личные опыты привели к тому, что эта книга написана именно о пакете Maple. Однако авторы свободны в своих увлечениях и готовы к новым опытам.

Приведем данные двух исследований, посвященных сравнению пакетов компьютерной алгебры и их оценке. Результаты первого исследования взяты из статьи в PC Magazine [11] (они распространяются с рекламным пакетом фирмы Waterloo Maple Inc.) и представлены на следующей диаграмме:



На этой и последующей гистограмме по оси абсцисс отложена относительная интегральная характеристика производительности каждого из пакетов. Результаты второго исследования взяты из рекламного набора фирмы Macsyma Inc. и показаны на диаграмме



Как и всякое сопоставление, эти сравнения условны, но хочется отметить, что если сложить столбики из двух диаграмм и разделить на два, то получится приятный для Maple результат.

11.6. Книги по Maple

Список книг, описывающих Maple и разнообразные приложения его для преподавания и исследований, содержит более 60 названий на английском языке, а также некоторое количество ссылок на

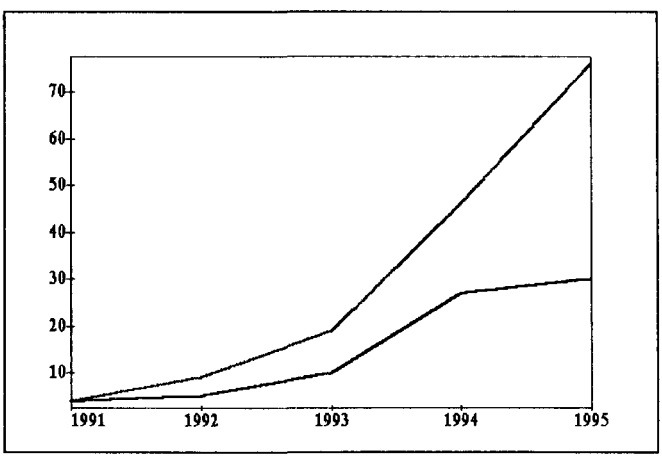
книги на немецком, французском, испанском, японском и, кажется, китайском языках.

Приведем данные о количестве книг по Maple, изданных в девяностые годы, на основе списка литературы, помещенного на WWW-странице Waterloo Maple Inc.

1991	1992	1993	1994	1995
4	5	10	27	30

Здесь авторы не удержались от того, чтобы не построить график числа книг, вышедших в каждый год, и общего количества за эти годы.

```
> a:=[1991,4,1992,5,1993,10,1994,27,1995,30]:
> s:=0: j:=0: for i in a do j:=j+1;
> if i>1990 then b[j]:=i else s:=s+i; b[j]:=s; fi;
> od: c:=convert(b,list):
> plot({a,c},x=1991..1995,axes=boxed);
```



11.7. Maple V – The Power Edition

В начале 1996 г. очередная реализация Maple V под названием The Power Edition вышла в свет. Сделаны шаги в развитие текстовых умений: появились сворачиваемые разделы, усовершенствовано управление стилями и фонтами. Теперь и вводимые команды и ре-

зультаты вычислений представляются в естественном математическом виде (знаки интегралов, дроби, матрицы т.д.).

Новая реализация дает возможность работы с несколькими документами одновременно. Это полезно при организации большого проекта, когда данные являются общими для всех документов, а также при создании обучающих систем с помощью Maple. Специальный MDI (Multiply-document interface) интерфейс обеспечивает установление гиперсвязей между отдельными документами. Имеется также возможность одновременной независимой работы с разными документами.

Улучшена справочная система. Добавлен режим поясняющего комментария при обходе пунктов меню или иконок (balloon help) и развиты системы просмотра общей гипертекстовой базы справки и поиска по ключевым словам.

Перечисленное улучшило интерфейс с тем, чтобы обеспечить пользователю работу в привычной обстановке программных сред общего назначения. Это техническое, но важное усовершенствование устраняет ряд анахронизмов, унаследованных от дографического интерфейса.

В части математических умений добавилась возможность решения систем неравенств при помощи многоцелевой команды **solve**. Расширены права кусочно-гладких функций, которые можно дифференцировать, интегрировать, а также использовать в дифференциальных уравнениях. Для целых классов уравнений в частных производных теперь выдаются аналитические решения. Усовершенствования коснулись команд матричной декомпозиции и возможностей проведения интегральных преобразований. Продвинуты темы специальных функций и вычисления бесселевых функций для комплексной плоскости. Есть улучшения среди численных методов решения дифференциальных уравнений.

Изменения графических возможностей включают переписанный пакет geometry, возможность изображения нескольких графиков для набора данных, новый пакет **plottools** для манипулирования составными объектами. Теперь доступна работа с изображениями и анимация непосредственно в тексте документа.

Для ожидающих изменений в языке программирования сообщим, что набор отладочных средств включает стандартные для обычных языков программирования точки останова (breapoints), про-

смотра (watch-points) и пошаговую отладку Maple-процедур. Для Windows реализован OLE2 сервис.

Значительно улучшены вывод в виде фортрановского или на языке C кода, а также превращение документа (worksheet) в текст стандарта LaTeX..

Все это потребовало увеличения минимального размера оперативной памяти. Для Windows-сред под Power Edition нужно теперь не менее 8 Мегабайт.

Проведенные сравнения Power Edition и Maple V Release 3 показали, что усовершенствование ряда команд и интерфейса повлекло замедление работы пакета и повысило требования к оперативной памяти и мощности процессора. Реально для работы требуется не менее 16 Мегабайт ОЗУ, так что надежная и быстрая Release 3 пока остается предпочтительной для многих пользователей.

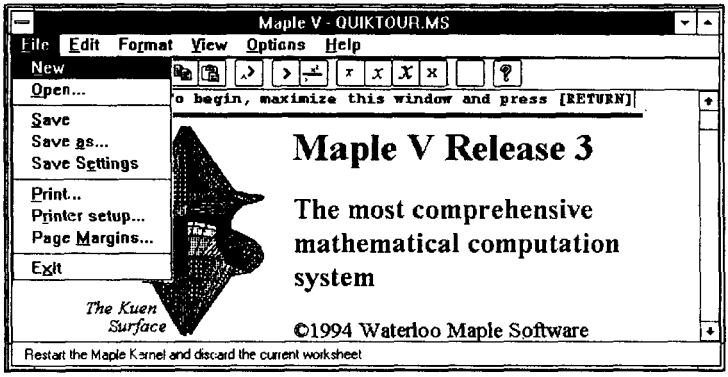
Список литературы

1. Darren Redfern. The Maple Handbook. Springer-Verlag, 1993. – 499 p.
2. B.W. Char, K.O. Geddes, G.H. Gonnet, B.L. Leong, M.B. Monagan, S. M .Watt. First Leaves: A Tutorial Introduction to Maple V. Springer-Verlag, 1992. – 253 p.
3. Демидович Б.П. Сборник задач и упражнений по математическому анализу. М.: Наука, 1977.
4. Задачи и упражнения по математическому анализу для втузов. Под редакцией Б.П. Демидовича. М.: Наука, 1970.
5. Лихтенберг А., Либерман М. Регулярная и стохастическая динамика. – М.: Мир, 1984. – 528 с.
6. Ахромеева Т.С., Курдюмов С.П., Малинецкий Г.Г., Самарский А. А. Нестационарные структуры и диффузионный хаос. – М.: Наука, 1992. – 544 с.
7. Лоренц Э. Детерминированное непериодическое течение. В книге: Странные аттракторы. – М.: Мир, 1981. – с.88–116.
8. Юдович В.И. Асимптотика предельных циклов системы Лоренца при больших числах Рэлея. Деп. ВИНИТИ N 2611-78.
9. Флетчер К. Численные методы на основе метода Галеркина. М.: Мир, 1988.
10. Simon B. Comparative CAS Reviews // Notices, 1992, **39** no.7, p.700–710.
11. Mathematical Genius // PC Magazine, October 1994.

12. Приложение

12.1. Пункты меню Windows-версии

При запуске Windows-версии Maple V Release 3 появляется типовое окно Windows. Читателю, знакомому с продуктом фирмы Microsoft, не нужно объяснять назначение многих пунктов меню и иконок, но для полноты картины коротко перечислим все пункты и все иконки.



Основное меню Maple состоит из шести пунктов: меню работы с файлами **File**, меню редактирования **Edit**, меню оформления документа **Format**, дополнительных меню **View** и **Options** с командами, управляющими функционированием рабочего поля, и меню справки **Help**.

Приведем описание пунктов меню. Рядом с названием пункта будем указывать эквивалентную по действию комбинацию клавиш. Многоточиями снабжены пункты, вызывающие дополнительные меню или запросы. Ряд пунктов является переключателями – задающими или отключающими некоторый режим. Включение переключателя отмечается в меню служебным символом '√'. Выбор пунктов меню осуществляется либо мышью, либо совместным нажатием клавиши Alt и клавиши с подчеркнутой буквой меню (**F** – для меню **File**, **E** – для меню **Edit** и т.д.).

Меню работы с файлами **File** содержит следующие подпункты:

New — Начало работы с новым файлом.
Open — Считывание файла с диска.
Save — Сохранение текущего файла.
Save as... — Сохранение файла под новым именем.
Save Settings — Сохранение установок Maple.
Print... — Печать документа.
Print setup... — Установка параметров печати.
Page Margins... — Установка параметров страницы.
Exit — Выход из программы.

Меню редактирования **Edit**:

Cut **Ctrl+X** — Удаление выделенного текста с сохранением в буфере.
Copy **Ctrl+C** — Копирование выделенного текста в буфер.
Paste **Ctrl+V** — Копирование текста из буфера, начиная с места, указанного курсором.
Delete Delete — Удаление выделенного текста.
Copy to... — Копирование содержимого буфера в файл.

Меню оформления документа **Format**:

Text Region F5 — Превращение строки ввода в текстовую строку и наоборот.
Split Group F3 — Разделение текущей строки на две.
Join Group F4 — Слияние текущей и предыдущей строки.
Insert Page Break — Вставка жесткого разделителя страниц.
Insert New Region — Добавление строки ввода:
Above Ctrl+O — перед текущей.
Below Ctrl+I — после текущей.
Remove All — Удаление:
Input — строки ввода,
Output — строки вывода,
Text — текстовой строки,
Graphics — графика.
Fonts — Назначение фонтов:
Input — ввода,
Output — вывода,
Text — текста.
Math Style — Определение размера математических символов:
Large — большой,
Medium — средний,
Small — маленький,
Character — задание шрифтов и размеров символов.
Execute Worksheet – Вычисление документа.

Меню **View**:

Separator Lines F9 — Показ разделителей (переключатель).
Input Prompts F10 — Показ приглашения ввода (переключатель).
Status Bar F2 — Показ окна статуса, информирущего об использованных Maple памяти и времени (переключатель).
Tool Bar — Показ иконок (переключатель).
Plot Tool Bars — показ иконок в графических окнах (переключатель).

Дополнительное меню **Options**:

Save Kernel State — Сохранение текущего (внутреннего) состояния сессии.
Confirmation Checks — Отказ от обязательного подтверждения ряда операций (выход, удаление).
Fast Graphics Redraw — Указание сохранять все трехмерные графики в памяти. Это полезно при работе с несколькими графиками и переключении с одного на другой.
Automatic Save Settings — Автоматическое сохранение текущего состояния.
Replace Mode — Переключатель режима ввода: вставка/замена.
Continuos mode — Режим перехода к следующей области экрана после выполнения текущей без вставки новой области.
Insert mode — Режим вставки новой области экрана после выполнения текущей.
Insert at end mode — Режим перехода к следующей области экрана после выполнения текущей с добавлением новой области в конце документа.

Меню справки **Help**:

Browser... F1 — Просмотр возможностей языка, команд и синтаксиса Maple. Многие команды имеют свои файлы справки, содержимое которых, в том числе примеры работы, может быть перенесено в рабочее поле.
Keyword search... Shift+F2 — Поиск по ключевым словам Maple.
Interface Help... Shift+F1 — Справка по интерфейсу Windows версии Maple.
General Maple Help... Ctrl+F1 — Общая справка.
About Maple V for Windows... — Информация о Maple для Windows.

Коротко опишем "Tool bar" – 16 иконок для выполнения часто используемых команд.

 – Создать новый документ.

 – Загрузить существующий документ.

 – Сохранить текущий документ.

 – Распечатать текущий документ, целиком или часть.

 – Вырезать выделенное и поместить в буфер.

 – Копировать выделенное в буфер.

 – Вставить содержимое буфера в документ.

 – Вставить приглашение ввода после текущей области.

 – Выдавать приглашение строки ввода (переключатель).

 – Выдавать разделители областей (переключатель).

 – Выдавать математические символы в стандартной нотации (малые буквы).

 – Выдавать математические символы в стандартной нотации (средние буквы).

 – Выдавать математические символы в стандартной нотации (большие буквы).

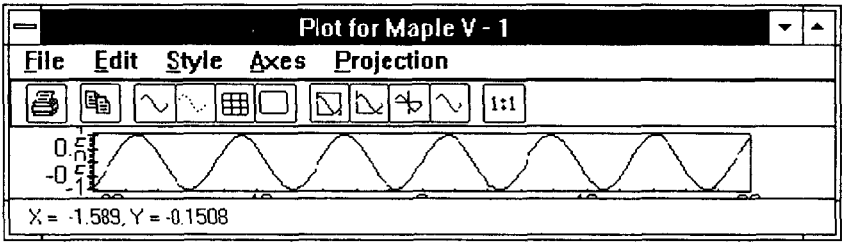
 – Выдавать математические символы обычными буквами.

 – Прервать вычисление.

 – Запустить Maple Help Browser (просмотр возможностей языка, команд и синтаксиса Maple).

12.2. Пункты меню двумерной графики

В Windows-версии это меню появляется как результат работы команд двумерной графики. Перечислим назначение всех пунктов меню и иконок.



Пункт меню **File** содержит следующие подпункты:

Print – печать рисунка;
Printer Setup – установка параметров печати;
Exit – выход из меню двумерной графики.

Пункт **Edit** состоит из одного подпункта:

Copy Ctrl+C – занесение рисунка в буфер обмена.

Подменю **Style** имеет следующие подпункты:

ToolBar – показ иконок меню (переключатель).
StatusBar – показ статуса работы (переключатель).
Line – рисунок выводится линиями.
Point – рисунок выводится точками.
Patch – заполнитель с разделительными линиями.
Patch w/o grid – заполнитель без разделительных линий.
Symbol – задание символа, которым помечаются точки:
Cross – крест.
Diamond – ромб.
Point – пиксель.

Circle – окружность.
Box – квадрат.
Default – знак умножения.

Line Style – определение стиля вывода линий:

[1] Solid – непрерывная линия.
[2] Dash – пунктирная линия.
[3] Dot – линия выводится точками.
[4] DashDot – пунктирная линия с чередованием черточка-точка.
[5] DashDotDot – пунктирная линия с чередованием черточка-точка-точка.
[0] Default – тип линии, установленный по умолчанию.

Line Width – задание толщины линий:

[1] Thin – тонкая линия.
[2] Medium – линия средней толщины.
[3] Thich – толстая линия.
[0] Default – установленная по умолчанию толщина линии.

Подменю **Axes** задает тип осей координат:

Boxed – оси в виде прямоугольника с нанесенными шкалами:
Framed – оси с центром в левом нижнем углу графического окна:
Normal – обычные оси координат:
None – вывод рисунка без осей координат.

Подменю **Projection** управляет типом масштаба рисунка:

Constrained – одинаковый масштаб по осям координат:
Unconstrained – масштаб выбирается по размеру окна графики.

Теперь расшифруем смысл иконок:

 – печать рисунка.

 – занесение рисунка в буфер обмена.

 – вывод рисунка линиями.

 – вывод рисунка точками.

 – заполнитель с разделяющими линиями.

 – заполнитель без разделяющих линий.

 – оси координат в виде прямоугольника.

 – оси координат с центром в левом нижнем углу рисунка.

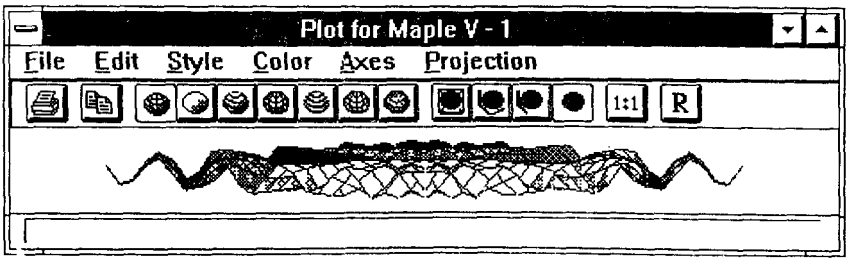
 – обычные оси координат.

 – рисунок выводится без осей координат.

– задание типа масштаба рисунка по осям координат (одинаковый по двум осям или выбирается по размеру окна графики).

12.3. Пункты меню трехмерной графики

В Windows-версии это меню появляется как результат работы команд трехмерной графики. Перечислим назначение всех пунктов меню и иконок.



Коротко опишем все его пункты и иконки:

Пункт **File** состоит из следующих подпунктов:

Print – печать рисунка;
Printer Setup – установка параметров печати;
Exit – выход из меню трехмерной графики.

Подменю редактирования **Edit**:

Copy **Ctrl+C** – копирование рисунка в буфер обмена;
Redraw Enter – повторный вывод рисунка на экран.

Подменю **Style** управляет представлением рисунка:

ToolBar – показ иконок меню (переключатель);
StatusBar – показ статуса работы внизу окна (переключатель);
Patch – заполнитель с разделительными линиями;
Patch w/o grid – заполнитель без соединительных линий;
Patch and contour – заполнитель с линиями уровня;
Hidden line – выводятся видимые соединительные линии без заполнителя;
Contour – выводятся только линии уровня;
Wireframe – выводятся все соединительные линии без заполнителя;
Point – изображение поверхности точками;
Symbol – задание символа, которым помечаются точки:
Cross – крест;
Diamond – ромб;
Point – пиксел;
Circle – окружность;
Box – квадрат;
Default – знак умножения;
Line Style – определение стиля вывода линий:
[1] Solid – непрерывная линия;
[2] Dash – пунктирная линия;
[3] Dot – линия выводится точками;
[4] DashDot – пунктирная линия с чередованием черточка–точка;
[5] DashDotDot – пунктирная линия с чередованием черточка-точка-точка;
[0] Default – *тип линии, установленный по умолчанию*;
Line Width – задание толщины линий:
[1] Thin – тонкая линия;
[2] Medium – линия средней толщины;
[3] Thich – толстая линия;
[0] Default – установленная по умолчанию толщина линии;
Grid Style – определение типа соединительных линий:
Grid Full – соединяются все соседние точки (триангуляция);

Grid Half – соединяются только точки, соседние по осям координат (прямоугольники).

В подменю **Color** определяется характер раскраски поверхности:

XYZ – цвет раскраски зависит от трех координат;
XY – цвет раскраски зависит только от горизонтальных координат;
Z – цвет раскраски зависит только от вертикальной координаты;
Z (Hue) – цвет раскраски зависит от вертикальной координаты; используется более яркая палитра;
Z (Greyscale) – цвет раскраски зависит от вертикальной координаты; используются оттенки серого цвета;
No Coloring – поверхность не раскрашивается;
No Lighting – рисунок выводится без подсветки;
User Lighting – подсветка, определенная пользователем;
Light Scheme 1 – подсветка по схеме 1;
Light Scheme 2 – подсветка по схеме 2;
Light Scheme 3 – подсветка по схеме 3;
Light Scheme 4 – подсветка по схеме 4;
Dither – рисунок выводится со штриховкой.

Подменю задания типа осей координат **Axes**:

Boxed – оси в виде прямоугольника с нанесенными шкалами;
Framed – оси с центром в левом нижнем углу графического окна;
Normal – обычные оси координат;
None – вывод рисунка без осей координат.

Подменю определения типа проекции **Projection**:

No Perspective – без перспективы;
Near Perspective – близкая перспектива;
Medium Perspective – средняя перспектива;
Far Perspective – дальняя перспектива;
Constrained – одинаковый масштаб по осям координат;
Unconstrained – масштаб выбирается по размеру окна графики.

Иконки меню трехмерной графики имеют следующий смысл:

 – печать рисунка;

 – занесение рисунка в буфер обмена;

 – при рисовании поверхности используется заполнитель и выводятся соединительные линии;

 – при рисовании поверхности используется только заполнитель;

 – при рисовании поверхности используется заполнитель и выводятся линии уровня;

 – при рисовании поверхности выводятся только видимые соединительные линии;

 – при рисовании поверхности выводятся только линии уровня;

 – при рисовании поверхности выводятся все соединительные линии;

 – поверхность выводится на экран точками;

 – оси координат выводятся в виде прямоугольника;

 – выводятся обрамляющие оси координат;

 – выводятся обычные оси координат (репер);

 – рисунок выводится без осей координат;

– задание типа масштаба рисунка по осям координат (одинаковый по двум осям / выбирается по размеру окна графики);

– повторный вывод рисунка на экран.

Алфавитный указатель

! 15
" 18
% 19
' 19
* 15
+ 15
- 15
-> 106
/ 15
: 17
, 17
^ 15
_C 19
_N 19
_Z 19
` 19

A

abs 20
add 62, 93
addcol 62
addedge 96
addrow 62
addvertex 96
adj 63
adjoint 63
algebraic 154
ALIGNABOVE 130
ALIGNBELOW 130
ALIGNLEFT 130
ALIGNRIGHT 130
ambientlight 132
angle 45, 66
animate 128
animate3d 139
antisymmetric 58, 86
appendto 110
arccos 21
arccosh 21
arccot 21
arccoth 21
arccsc 21
arccsch 21
arcsecs 21
arcsech 21
arcsin 21
arcsinh 21
arctan 21
arctanh 21
are_collinear 41, 44
are_concurrent 41, 44
are_harmonic 41
are_orthogonal 41
are_parallel 41, 44
are_perpendicular 41, 44
are_similar 41, 44
are_tangent 41, 44
area 42, 45
array 14, 32, 58
arrows 122
assign 18, 37
associative 86
assume 47
asympt 56
axes 121, 131

B

basis 66
bequal 95
BesselI 21
BesselJ 7, 21
BesselK 21
BesselY 21
Beta 21

bezout 68
binary 86
binomial 99
bisector 42
blockmatrix 60
bsimp 95
by 104

C

C 112
canon 95
center 42, 45
centroid 42
cfracpol 90
changevar 54
charpoly 64
chebpade 97
chebyshev 56
choose 100
circle 41, 42
circumcircle 42
close 111
coeff 28, 35
coeffs 35
coeftayl 56
col 61
coldim 60
collect 25, 116
color 121, 132
colorstyle 128
colspace 66
combinat 100
combine 26
commutative 86
complex 39
compose 93
concat 61
concyclic 42
cond 63
confracform 97
conjugate 31
CONTOUR 131
contourplot 137
convert 7, 29, 30, 36, 52, 60
convexhull 42
coordinates 42, 45
coords 121, 125, 131
coplanar 45
copyinto 61
cos 20
cosh 21
cot 20
coth 21
crossprod 67
csc 20
csch 21
curl 67
CURVES 130
cylinderplot 137

D

D 52, 70
Dchangevar 77, 84
define 85, 86
definite 65
degree 35
delcols 61
delrows 61
denom 25, 26
densityplot 126, 173
DEplot 76, 77
DEplot1 77, 81
DEplot2 77, 81
describe 91
DESol 69, 74

Det 63
det 63
detailf 42
determine 101
DEtools 69, 73, 76, 77, 157
dfieldplot 77, 83
diagonal 58
diameter 42
Diff 51
diff 51, 70, 156
difforms 101
Digits 13, 32, 53
Dirac 21, 87
discont 50
discrim 35
display 92, 128, 155, 169, 173
display3d 139, 168
distance 42, 44, 45
distrib 95
distributions 92
diverge 67
divide 34, 35
do 104
dodecahedron 138
dotprod 67
Doubleint 54
dsolve 8, 69, 71, 159, 163, 164
dual 95

E

E 13
edges 97
Eigenvals 64
eigenvals 64
eigenvects 64
ellipse 43
else 103
end 107
enlif 104
environ 95
erf 21
ERROR 109, 153
eval 18, 31, 59
evalc 31
evalf 31
evalf(int) 53
evalhf 31
evalm 31, 62, 63
evalp 91
evalpow 93
example 23
exp 20
expand 24, 25, 26, 34
extend 61
extrema 50

F

factor 24, 26, 35, 145
false 13, 41
feasible 94
ffgausselim 65
FFT 87, 166
fi 103
fibonacci 100
fieldplot 122, 127
fieldplot3d 137
find_angle 43
fit 91
Float 74
font 122
for 104
forall 87
forget 101

fortran 111
fourier 87
frac 31
frame 128, 139
from 104
fsolve 38, 148

G

G 89
GAMMA 21
gamma 13
Gauss 102
gausselim 65, 127
GaussIn 89
gaussjord 65
gcd 35
genfunc 102
genmatrix 68
geom3d 40, 44
geometry 40, 41
GF 102
GInearest 91
GIprime 91
global 108
grad 67
gradplot 122, 127
gradplot3d 137
GramSchmidt 66
GRID 132
grid 122, 126, 132
grobner 102
Group 12, 85
group 101

H

H 89
Heaviside 21, 87
heights 137
help 22
hermite 65
hexahedron 138
HIDDEN 131
hilbert 60
histogram 142
horizontal 130
hornerform 98
htranspose 63

I

I 13
icosahedron 138
identity 58, 86
if 103
ifactor 6
iFFT 88
Im 31
implicitplot 122, 126, 173
implicitplot3d 137
importdata 91
in 105
incircle 43
indexed 14
infinity 13
infnorm 98, 99
inifunction 21
init 101
Input Region 12
insequence 128, 139, 169
Int 7, 52, 176
int 52, 171
Int [student] 54
inter 43, 45
interface 116, 120
interp 88
intersect 15

intparts 54
inverse 63, 86, 93
inversion 43
invfourier 88
invlaplace 88
is_equilateral 42
is_right 42
iscont 50
isolate 26
isolve 39
isprime 90

J

jacobian 67
jordan 65

K

kernel 65

L

L 89
labels 122, 126, 132
laplace 72
laplacian 67
lasterror 117
latex 112
laurent 56, 98
lcoeff 35
ldegree 35
leastsqrs 66
leastsquare 92
leftbox 141
LegendreE 21
LegendreEc 21
LegendreF 21
LegendreKc 21
LegendrePi 21
LegendrePic 21
length 32
lhs 26
liesymm 101
light 132
Limit 48
limit 48
linalg 16, 58, 64, 127
LINE 131
line 44
line3d 44
Linear 85
Lineint 54
linestyle 122
linsolve 66
list 32
listlist 132
ln 20
local 108, 153
log 20
log10 20
log[a] 20
logic 95
loglogplot 125
logp 91
logplot 125
lprint 111

M

makehelp 114
march 115
matrix 58
matrixplot 137
max 32, 33
maximize 50, 94
mellin 88
MESH 133

middlebox 141
midpoint 43, 45
min 32
minimax 98, 99
minimize 50, 94
minor 61
mint.exe 118
minus 15
msolve 39
mtaylor 56
mulcol 62
mulrow 62
multiply 62, 93

N

negative 93
Networks 96
nextprime 90
nops 27
norm 67
normal 25, 26
normalize 66
NPspinor 102
numapprox 56, 97
numbcomb 100
numbperm 100
numer 25, 26
numeric 72, 163
numpoints 121, 125, 132
numtheory 89

O

octahedron 138
od 104
odeplot 73, 76, 128, 137, 163
on_circle 42
on_line 42
on_plane 45
on_sphere 45, 46
op 27, 59
open 111
operator 108
options 108
Order 13, 32, 55, 56, 72
order 32
ordering 101
orhtopoly 89
orientation 131
orthog 66
orthopoly 8, 36
output 163
Output Region 12

P

P 89
pade 98
padic 89, 91
parallel 43, 45
PATCH 131
PATCHCONTOUR 131
PDEplot 77
permute 100
perpen_bisector 43
perpendicular 43, 45
phaseportrait 77, 83, 157
Pi 13
plane 44
PLOT 130, 168
plot 120, 123, 125
plot3d 134, 136, 137
plotdevice 116, 120
plotoutput 116, 120
plots 119, 163, 173
plottools 184

POINT 131
point 41, 44
point3d 44, 46
pointplot 137
POINTS 129
poisson 56
polar 31
polarplot 126
poligonplot 125
polygonplot3d 137
POLYGONS 130, 168
polyhedraplot 137
polyscale 137
polytype 138
powcreate 93
powdiff 93
powexp 93
powint 93
powlog 93
powpoly 93
powseries 93
powsolve 94
prettyprint 116
print 111
printf 111
printlevel 115
proc 107, 149, 150, 153, 169, 172
Product 49
product 49
projection 43, 45, 131, 138
projgeom 40
proot 36
psqrt 36

Q

quo 35
quotient 93

R

radius 43, 45, 46
rand 142
randbool 96
randmatrix 60, 127
random 91
randpoint 43
randpoly 36
rank 63
Re 31
read 110
readdata 113
readlib 15, 47, 112
readline 113
readshare 143
readstat 113
realroot 36
reflect 43, 45
related 23
rem 36
remember 108
remez 98
replot 128
residue 51
resolution 121
RETURN 109
reversion 93
rhs 8, 26, 147, 160
rightbox 141
RootOf 38, 71
round 31
row 61
rowdim 60
rowspace 66
rsolve 39

S

satisfy 96
save 110
scaling 121, 125, 131
scatter2d 92, 142
sec 20
sech 21
Separator 12
series 30, 32, 55, 71, 75
session 12
set 14, 32
setoptions 121
setoptions3d 132, 139
shading 132
share 143
showtangent 142
showtime 118
sides 43
signum 20
simplex 94
simplify 24, 25, 26, 28, 146
sin 20
singular 51
sinh 21
solve 6, 17, 20, 36, 66, 146
sort 32
spacecurve 138, 168
sphere 44, 45, 46
sphereplot 139
spline 88, 165
sqrt 20
stack 61
statevalf 91
statplots 91, 92
stats 16, 91, 113, 142
statsplot 142
string 13, 14, 154
student 53, 58, 141
style 121, 125, 131
submatrix 61
subs 7, 24, 26, 156, 157
subsop 27
subtract 93
subvector 61
Sum 48, 170
sum 19, 48
surfdata 139
swapcol 62
swaprow 62
sylvester 68
symbol 122, 125
symmetric 43, 45, 58, 86

T

T 89
tan 20
tangent 43, 45
tangentpc 43
tanh 21
tassume 101
tautology 96
taylor 7, 55
terminal 110
tetrahedron 45, 138
TEXT 114, 130
Text Region 12
textplot 126, 154
textplot3d 139
thickness 121
tickmarks 132
time 118
tis 101
title 121, 131
titlefont 124
to 104

totorder 101
tpsform 93
trace 16, 63, 115
transform 91
transpose 63
traperror 117
triangle 44
triangle3d 45
trigsubs 26
Tripleint 54
true 13, 41
trunc 31
tubeplot 139
type 28, 154

U

U 89
unapply 99, 107, 145
unary 86
unassign 37
union 15
untrace 115
updates 22
usage 23

V

value 47, 48, 164
vectdim 60
vector 58
verboseproc 116
vertical 130
vertices 97
view 131
volume 45, 46

W

whattype 15, 28
while 104
WIREFRAME 131
with 16, 47, 114, 119
words 118
worksheet 6
write 111
writeln 111
writeto 110

X

xtickmarks 121

Y

ytickmarks 121

Z

zero 86
Zeta 21

Оглавление

Введение 5
0.1. Первые команды 6
0.2. Структура книги 9
0.3. Благодарности 11
1. Среда Maple 12
1.1. Объекты 13
1.2. Типы переменных 14
1.3. Выражения 15
1.4. Команды Maple 15
1.5. Синтаксис 17
1.6. Стандартные функции 20
1.7. Справка 22
2. Аналитические преобразования 24
2.1. Операции с формулами 24
2.2. Преобразования типов 28
2.3. Операции оценивания 30
3. Элементарная математика 34
3.1. Операции с полиномами 34
3.2. Решение уравнений и неравенств 36
3.3. Геометрические пакеты 40
3.4. Планиметрия 41
3.5. Стереометрия 44
4. Математический анализ 47
4.1. Пределы, суммы, ряды 47
4.2. Исследование функций 49
4.3. Дифференцирование и интегрирование 51
4.4. Разложение и приближение функций 55
5. Линейная алгебра 58
5.1. Работа со структурой матрицы и вектора 60
5.2. Основные матричные и векторные операции 62
5.3. Решение задач линейной алгебры 63
5.4. Векторный анализ 67
6. Дифференциальные уравнения 69
6.1. Точные и приближенные решения 69
6.2. Численные решения 72
6.3. Структура DESol 74
6.4. Пакет DEtools 76
7. Математические библиотеки 85
7.1. Определение абстрактных операторов 85
7.2. Интегральные преобразования 87
7.3. Интерполяция 88
7.4. Ортогональные полиномы 89
7.5. Теория чисел 89
7.6. Статистика 91

7.7. Степенные разложения ... 93
7.8. Линейная оптимизация ... 94
7.9. Математическая логика ... 95
7.10. Теория графов ... 96
7.11. Аппроксимация функций ... 98
7.12. Комбинаторика ... 99
7.13. Группы и формы ... 101
7.14. Другие пакеты ... 101
8.Программирование ... 103
8.1. Условный оператор ... 103
8.2. Операторы цикла ... 104
8.3. Процедуры-функции ... 106
8.4. Процедуры ... 107
8.5. Команды ввода/вывода ... 110
8.6. Создание собственных библиотек ... 113
8.7. Отладка программ ... 115
9. Графика в Maple ... 119
9.1. Опции двумерной графики ... 120
9.2. Команды двумерной графики ... 123
9.3. Двумерные графические структуры ... 129
9.4. Опции трехмерной графики ... 131
9.5. Структуры трехмерной графики ... 132
9.6. Команды трехмерной графики ... 134
9.7. Иллюстративные графические команды ... 141
10. Мини-исследования ... 144
10.1. Логистическое отображение ... 144
10.2.Разложение функции в ряд Фурье ... 153
10.3.Система Лоренца как маятник с обратной связью ... 155
10.4. Течение вязкой жидкости в канале ... 170
11. Заключение ... 178
11.1. Maple на марше ... 178
11.2. Maple и университеты ... 180
11.3. Maple и периодика ... 180
11.4. Maple и Internet ... 180
11.5. Maple и другие пакеты ... 181
11.6. Книги по Maple ... 182
11.7. Maple V – The Power Edition ... 183
Список литературы ... 186
12. Приложение ... 187
12.1. Пункты меню Windows-версии ... 187
12.2. Пункты меню двумерной графики ... 191
12.3. Пункты меню трехмерной графики ... 193
Алфавитный указатель ... 197