Б.М.Манзон

MAPLE V POWER EDITION

Издание является кратким руководством пользователя универсального математического пакета *Maple V Power Edition* (версия 4), широко используемого как для преподавания математики, так и для профессиональной работы. Пакет позволяет решать численно и аналитически большое количество математических задач любого уровня сложности. Благодаря встроенным алгоритмам многие задачи в *Maple V* решаются методом простых команд.

В книге на примерах из различных областей математики описаны методы проведения аналитических и численных расчетов и оформления выполненной работы для публикации. Описан также язык программирования *Maple* и методы создания при помощи него команд и функций, расширяющих функциональность пакета.

Книга будет полезна всем, кто изучает математику или использует ее в своей работе: от студентов и школьников, преподавателей средней и высшей школы до научных и инженерных работников.

Содержание

1. ЧТО ТАКОЕ MAPLE V **8**
2. БЫСТРЫЙ СТАРТ **10**
3. ИНТЕРФЕЙС **13**
4. ОБЪЕКТЫ MAPLE **17**
4.1. Язык программы 17
4.2. Структура объектов 18
Выражения 18
Числа и константы, строки и имена 19
1. Целые и рациональные числа 19
2. Математические константы 20
3. Смешивание и совместимость различных типов констант 20
4. Строки 20
5. Имена 21
6. Оператор конкатенации 23
7. Использование кавычек в Maple 23
Последовательности выражений 25
Наборы и списки 26
1. Наборы 26
2. Оперирование элементами набора (команды union, intersect,minus) 27
3 Списки 27
4. Оперирование элементами списка (команды select, remove, zip, sort) 28
Операторы присваивания и уравнения 30
Функции 32
Операторы Maple 35
1. Оператор композиции 35

2. Нейтральный оператор 36
4.3. Определение типов объектов 36
4.4. Анализ структуры объектов 38
5. КОМАНДЫ MAPLE 40
5.1. Последовательности параметров 40
5.2. Как вызвать команду? 41
Автоматически загружаемые и загружаемые из библиотек команды 42
Команды в пакетах 42
5.3. Некоторые часто используемые команды 43
Преобразование выражений 43
Части выражения (команды lhs, rhs, numer, denom, remove, has, select, indet, subs, subsop) 44
Команда simplify 47
Команды expand и factor 48
Команда normal 49
Команда combine 49
Команда assume 49
Команды map, add, mul 51
Изменение типа выражения (команда convert) 54
6. ПРИМЕРЫ ВЫЧИСЛЕНИЙ 56
6.1. Преобразование алгебраических выражений 56
Многочлены и рациональные дроби 56
Сложные радикалы 57
Тригонометрические выражения 57
6.2. Решение уравнений и неравенств 58
Решение систем уравнений 58
Системы линейных уравнений 60
Корни многочленов 62
Системы нелинейных уравнений 64
Решение рекуррентных и функциональных уравнений 65
Решение трансцендентных уравнений и систем 66
Решение тригонометрических уравнений 66
Решение неравенств 67
6.3. Нахождение экстремумов функций, симплекс-метод 68
6.4. Дифференцирование 69
6.5. Пределы 72
6.6. Интегрирование 73
Аналитическое интегрирование 73
Численное интегрирование 75
6.7. Суммы и произведения 76
6.8. Примеры из линейной алгебры 77
Массивы 77
Специальные типы матриц 78

Управление элементами массивов 78
Команды пакета linalg 80
6.9. Обыкновенные дифференциальные уравнения 83
6.10. Уравнения в частных производных 89
7. ГРАФИКИ И АНИМАЦИЯ В MAPLE 97
7.1. Двухмерные графики 97
Графики, построенные при помощи команды plot 98
Графики, построенные при помощи команд пакета plots 106
Графика пакета plottools 119
Графика статистического пакета 120
Графика пакета DEtools 123
Графика геометрического пакета 127
7.2. Трехмерные графики и трехмерная анимация 129
Графики команды plot3d 129
Построение трехмерных графиков с помощью команд пакета plots 134
Графика пакета DEtools 145
Графика пакета plottools 147
Трехмерная анимация 149
8. ПРОГРАММИРОВАНИЕ В СРЕДЕ MAPLE 150
8.1 Процедурное программирование 150
8.1.1. Базисные конструкции языка 150
If/then/else/fi 150
lf/then/elif/then/.../else/fi 151
for/f rom/by/to/do/od 151
While/do/od 151
8.1.2. Процедуры 152
Параметры процедуры 155
Переменные операционной среды 157
Команда прерывания ERROR 158
Рекурсивные процедуры, команда RETURN, опция remember 159
Вложенные процедуры 161
Ньютоновская итерация 164
Оператор аффинного преобразования 166
8.1.3. Методы отладки программ 170
Трассировка 170
Отладчик 173
Чтение кодов библиотечных процедур 175
8.1.4. Сохранение процедур и чтение их в сеансе Maple 176
8.1.5. Создание собственной библиотеки и оформление справки по ее командам 176
8.1.6. Чтение и запись данных в файлы 180
Запись данных в файл 180
Чтение данных из файла 181

8.1.7. Перекодировка процедур на языки Си и Фортран 185
8.2. Программирование свойств и правил вычисления функций и операторов 187
8.2.1. Команда define 187
8.2.2. Программирование правил вычисления 190
8.2.3. Сравнение с шаблоном 192
8.3. Пакет Domains 194
8.3.1. Домены в Domains 194
8.3.2. Примеры использования пакета Domains 195
8.3.3. Пакет Domains в интерактивном режиме 202
9. СПЕЦИАЛИЗИРОВАННЫЕ ПАКЕТЫ MAPLE 204
9.1 DEtools — пакет дополнительных средств для дифференциальных уравнений 204
9.2. Domains — пакет для разработки кодов сложных алгоритмов 204
9.3. GF — пакет "поля Галуа" 205
9.4. Gausslnt — пакет Гауссовых целых чисел 205
9.5. LREtools — пакет для проведения расчетов с рекуррентными соотношениями 206
9.6. combinat — пакет комбинаторики 207
9.7. combstruct — пакет комбинаторных структур 207
9.8. difforms — пакет дифференциальных форм 208
9.9. finance — пакет финансовой математики 208
9.10. genfunc — пакет для проведения расчетов с производящими функциями 209
9.11. geometry — геометрический пакет 209
9.12. grobner — пакет процедур для нахождения базиса Гробнера 209
9.13. group — пакет групп перестановок и конечно-представимых групп 210
9.14. inttrans — пакет интегральных преобразований 211
9.15. liesymm — пакет симметрии Ли 211
9.16. linalg — пакет линейной алгебры 213
9.17. logic — пакет математической логики 214
9.18. networks — пакет теории графов 215
9.19. numapprox — пакет численной аппроксимации функций 217
9.20. numtheory — пакет теории чисел 217
9.21 orthopoly — пакет ортогональных полиномов 218
9.22. padic — пакет для оперирования р-адическими числами 218
9.23. plots — пакет команд графики и анимации 219
9.24. plottools — пакет вспомогательных инструментариев графики 219
9.25. powseries — пакет генерации и преобразования степенных рядов 220
9.26. simplex — пакет линейной оптимизации 221
9.27. stats — пакет статистики 221
9.28. student — пакет для изучения математики и программирования 224
9.29. sumtools — пакет для вычислений конечных и бесконечных сумм 224

9.30. tensor — пакет тензорной алгебры 225
9.31. totorder — пакет полного упорядочения имен 230
9.32. Библиотека совместного пользования (share-библиотека) 231
ЗАКЛЮЧЕНИЕ 233
ЛИТЕРАТУРА 234
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 235

Предметный указатель

about 49
act 226
acycpoly 215
add 51
addcol 213
addcoords 97
addedge 215
additionally 91
addrow 213
addvertex 215
adjacency 215
adjoint 213
Algebra 231
allpairs 215
allvalues 123
altitudes 127
ambientlight 129
Analysis 231
ancestor 215
and 214
angle 213
animate 219
animate3d 219
annul 211
anova 222
antisymmetrize 226
arc 219
AreCollinear 128
args 156
array 78
arrivals 215
arrow 219
arrows 109
assign 31
assume 47
augment 213
autosimp 211
axes 223

backsub 213
balloone help 14
band 213
basis 221
bell 207
bequal 214
bernoulli 217
bezout 213
bicomponents 215
bigomega 217
binomial 207
bipolarcylindrical 134
bispherical 134
blockmatrix 213
bsimp 214
by 151

C 229
Calculus 231
canon 214
cardiodal 134
cardiodcylindrical 134
cartprod 207
casscylindrical 134
Catalan 20
center 210
centralizer 210
centroid 127
change_basis 226
changecoords 219
changevar 224
character 207
charmat 213

charpoly 213
chebdeg 217
chebmult 217
chebpade 217
chebsort 217
chebyshev 217
Chi 207
cholesky 213
choose 207
Christoffel 229
Christoffell 225
Christoffel2 225
chrompoly 215
circle 219
circumcircle 127
close 211
coefficientofVariation 182
col 213
coldim 213
color 97
colspace 213
colspan 213
combinat 207
Combinatorics 231
combine 224
combstruct 207
commutator 226
companion 213
compare 226
complement 215
complete 215
completesquare 224
complex 72
complexplot 219
complexplot3d 219
components 215
composition 207
cond 213
confocalellip 134
confocalparab 134
conformal 219
confracform 217
conic 209
conical 134
conj 226
conjpart 207
connect 215
connectivity 215
connexF 226
const 208
constcoeffsol 206
Contents 16
continuous 74
contourplot 219
contourplot3d 219
contours 107
contract 215, 226
Conversions 231
convert 210
convert/frominert 214
convert/MOD2 214
convert/toinert 214
convertNP 226
convexhull 221
coordplot 219
coordplot3d 219
coordplots 116
copyinto 213
core 210
cosets 210
cosrep 210
count 207
countcuts 215
countmissing 182
counttrees 215
Courses 231
cov_diff 226
covariance 182
create 226
crossprod 213
cterm 221
cube 215
curl 213
cycle 215
cyclebase 215
cylinderplot 219
cylindrical 134

d 208, 211
D 224
dl metric 226
d2metric 226
daughter 215
Dchangevar 204
decile 182
decodepart 207
deiform 208
defme_zero 221
definite 213
degreeseq 215
delcols 213
delete 215
delrows 213
delta 206
denom 44
DenseUnivariate Polynomial 194
densityplot 219
departures 215
DEplot 204
DEplot3d 204
depvars 211
derived 210
DerivedS 210
describe 222
del 213
determine 211
DEtools 204
dfieldplot 126
diag 213
diameter 215
Diff 224
Diff 40
diff 40, 41
difforms 208
Digits 87
dinic 215
Dirac 46
directional_diff 226
discont 101
disk 219
dispersion 206
display 219, 221
display_allGR 226
display3d 219
displayOR 226
distance 224
distrib 214
ditto 154
diverge 213
divisors 217
djspantree 215
do 151
dodecahedron 215, 219
Domains 204
done 173
dotprod 213
Doubleint 224
draw 207,215
dsegment 209
dsolve 31
dual 214,221,227
duplicate 215
dvalue 211

edges 215
eigenval 213
eigenvals 81
eigenvect 213
eighbors 215
Einstein 226
elif 151
ellipse 209, 219
ellipsoidal 134
ellipticArc 219
EllipticK 75
else 150
encodepart 207
end 152
ends 215
Engineering 231
Enter 10
entermatrix 213
entermetric 227
environ 214
equal 213
equality 221

equate 224
ERROR 152, 158
Eta 211
euler 217
eval 154
evalb 30
evalf 46
evalm 78
evaln 25
evalp 218
evalpow 220
eweight 215
exp(l) 26
expansion 218
exponential 213
extend 213
extended_gosper 224
exterior_diff 227
exterior_prod 227
extrema 224
extvars 211

F 217
factor 48
factorEQ 217
factorset 217
FALSE 124
false 225
feasible 221
fermat 217
ffgausselim 213
fi 150
fibonacci 207, 213
fieldplot 219
fieldplot3d 219
finance 208
finduni 210
finished 207
finite 210
firstpart 207
fit 222
Float 19
float 36
flow 215
flowpoly 215
for 151
forget 230
form 208
formpart 208
fortran 187
forwardsub 213
fourier 211
fouriercos 211
fouriersin 211
fraction 36
frame 227
frames 149
FresnelC 104
frobenius 213
from 151
function 218
fundcyc 215

gamma 20
gausselim 213
Gausslnt 205
gaussjord 213
gbasis 210
geneqns 213
genuine 209
genmatrix 213
geodesic_eqns 227
geometricmean 182
geometry 209
Geometry 231
get_char 227
get_compts 227
get_rank 227
getcoeff 211
getform 211
getlabel 215
GF 205
GIbasis 205
Glchrem 205
Gldivisor 205
GIfacpoly 205
GIfacset 205
Glfactor 205

Glfactors 205
Glgcd 205
GIgcdex 205
Glhermite 205
Glissqr 205
Gllcm 205
GImcmbine 205
GI nearest 205
GInodiv 205
GInorm 205
GInormal 205
Glorder 205
GIphi 205
GIprime 205
Glquadres 205
Glquo 205
GIrem 205
GI roots 205
girth 215
GIsieve 205
GIsmith 205
GIsqrfree 205
GIsqrt 205
Glunitnormal 205
global 152
gosper 224
grad 213
gradplot 219
gradplot3d 219
graph 215
graphical 215
Graphics 231
graycode 207
grelgroup 210
grid 95
grobner 209
group 210
groupmember 210
grouporder 210
gsimp 215
gsolve 210
gunion 215

hadamard 213
hankel 211
harmonicmean 182
has 37
hasclosure 211
hastype 37
head 215
hemisphere 219
hermite 213
hessian 214
hexahedron 219
hilbert 211, 214
histogram 120
hook 211
hornerform 217
htranspose 214
hyperbola 209, 219
hypercylindrical 134
hypergeomsols 206
hyperrecursion 224
Hypersum 224
hypersum 224
hyperterm 224

I 20
icosahedron 215, 219
identity 78
if 150, 151
ifactor 217
ifactors 217
ihermite 214
imagunit 217
implicitplot 219
implicitplotSd 219
importdata 221
incidence 215
incident 215
indegree 215
indepvars 211
indet 44
index 217
indexed 22
indexfunc 214
induce 215
inequal 219

infinity 20
infnorm 217
infolevel 135
inits 123
innerprod 214
Int 224
intbasis 214
integer 36
integrand 224
inter 210
intercept 224
interface 175
intersect 27
intparts 224
inttovec 207
inttovec 43
inttrans 211
invars 227
invcasscylindrical 134
invcfrac 217
invellcylindrical 134
inverse 214, 220
invert 227
invfourier 211
invfunc 36
invhilbert 211
invlaplace 211
invoblspheroidal 134
invperm 210
invphi 217
invprospheroidal 134
is 28
isabelian 210
ismith 214
isnormal 210
isolate 224
isolve 217
isplanar 215
isprime 217
issimilar 214
issqrfree 217
issubgroup 210
iszero 214
iterstructs 207
ithprime 217

J 217
jacobi 217
jacobian 214
Jacobian 226
Jordan 214
JordanBlock 213

kernel 214
Killing_eqns 226
kronecker 217
kurtosis 182

L 217
labels 143
lambda 217
laplace 211
laplacian 214
lastpart 207
laurent 217
lcoeffp 218
LCS 210
leadmon 210
leastsqrs 214
left 72
leftbox 224
leftsum 224
legendre 217
length 21
Levi_Civita 226
lhs 44
libname 179
Lie 211
Lie_diff 226
liesymm 211
light 129
Limit 224
limit 72
lin_com 227
linalg 213
line 209, 219
linearcorrelation 182
linecolor 126

Lineint 224
linestyle 119
linsolve 214
listcontplot 219
listcontplotSd 219
listdensityplot 219
listplot 219
listplotSd 219
local 71
logcoshcylindrical 134
logcylindrical 134
logic 214
loglogplot 219
logplot 219
lower 227
Lrank 211
LREtools 206
LUdecomp 213

makeforms 211
makeproc 224
map 51
map2 52
matadd 214
matrix 214
matrixplot 219
max 26
maxdegree 215
maximize 221, 224
maxwellcylindrical 134
mcombine 217
mean 41
meandeviation 182
median 182
mersenne 217
method 83
mgear 84
middlebox 224
middlesum 224
midpoint 224
mincut 215
mindegree 215
minimax 217
minimize 221, 224
minkowski 217
minor 214
minpoly 214
minstep 87
minus 27
mipolys 217
mixpar 208, 211
mlog 217
mobius 217
mode 182
moment 182
mroot 217
msqrt 217
mul 51
mulcol 214
mulperms 210
multconst 220
multinomial 207
multiply 214, 220

nearestp 217
networks 215
new 215
nextpart 207
nextprime 217
nops 27, 38
norm 214
NormalClosure 210
normald 222
normalf 210
normalize 214
normalize r 210
not 214
notchedbox 120
npcurve 227
npspin 227
nthconver 217
nthdenom 217
nthnumer 217
nthpow 217
numapprox 217
numbcomb 207
numbcomp . 207
Number Theory 231

numbpart 207
numbperm 207
numer 44
numeric 28
Numerics 231
numpoints 97
numtheory 217

oblatespheroidal 134
octahedron 215, 219
od 151
odeplot 219
op 28
options 207
optionsclosed 112
optionsexcluded 112
optionsopen 112
or 214
orbit 210
order 217
ordering 230
orderp 218
ordp 218
orientation 129
orthocenter 127
orthopoly 218
outdegree 215
output 86

padic 218
parabola 209
paraboloidal 134
paraboloida!2 134
paracyiindrical 134
pareto 219
parity 208
partial_diff 227
partition 207
path 215
PDEplot 204
pdesolve 89
pdexpand 217
percentile 182
permanent 214
permgroup 210
permrep 210
permute 207
permute_indices 227
petersen 215
petrov 227
phaseportrait 126
phi 217
piecewise 95
pieslice 219
pivot 214, 221
pivoteqn 221
pivotvar 221
plot 13
plot3d 11
plots 219
plottools 219
point 209, 219
Point 224
pointplot 219
pointplot3d 219
polarplot 219
polygon 219
polygonplot 219
polygonplot3d 219
polyhedraplot 2""
potential 214
powadd 220
powcos 220
powcreate 220
powdiff 220
powerset 207
powexp 220
powint 220
powlog 220
powpoly 220
powseries 220
powsin 220
powsolve 220
powsqrt 220
powsubs 224
pprimroot 217
precision 185
pres 210

prevpart 207
prevprime 217
primroot 217
print 151
printlevel 157
proc 65
prod 227
Product 224
product 24
Programming 231
prolatespheroidal 134
prolong 211
protect 30

QRdecomp 213
quadraticmean 182
quantile 120
quantile2 120
quartile 182
quit 173
quotient 220

radical 65
radnormal 75
raise 227
randbool 214
randcomb 207
randmatrix 214
random 215, 222
randpart 207
randpart 43
randperm 207
randperm 43
randpoly 62
randvector 214
range 176
rank 214, 215
rankpoly 215
ratio 221
ratpolysols 206
ratvaluep 218
read 176
readlib 205
real 72
REcontent 206
REcreate 206
rectangle 219
reduce 211
references 214
remember 159
remez 217
remove 28
REplot 206
replot 219
REprimpart 206
REreduceorder 206
restart 46
REtoDE 206
REtodelta 206
REtoproc 206
RETURN 152
reversion 220
rgf_charseq 209
rgf_encode 209
rgf_expand 209
rgf_findrecur 209
rgfjiybrid 209
rgf_norm 209
rgfjjfrac 209
rgf_relate 209
rgf_sequence 209
rgf_simp 209
rgf_term 209
rhs 44
Ricci 226
Ricciscalar 226
Riemann 226
RiemannF 226
right 72
rightbox 224
rightsum 224
rootlocus 219
RootOf 62
rootp 218
roots 49
rootsunity 217
rosecylindrical 134
rotate 219

row 214
rowdim 214
rowspace 214
rowspan 214
rsolve 206

safeprime 217
save 176
scalar 208
scalarmul 214
scalarpart 208
scale 219
scatter Id 120
scatter2d 120
scene 123
segment 209
select 28
semilogplot 219
semitorus 219
seq 23
series 54
setoptions 219
setoptions3d 219
setup 211, 212, 221
share 231
shift 206
shortpathtree 215
show 205, 215
showtangent 224
shrink 215
siderel 47
sigma 217
signum 47
simpcomb 224
simpform 208
simplex 221
simplify 47
simpson 224
singval 214
sixsphere 134
skewness 182
slope 224
smith 214
solvable 210
solve 31
sort 28
spacecurve 219
span 215
spanpoly 215
spantree 215
sparse 78
sparsematrixplot 219
specification 207
sphere 219
sphereplot 219
spherical 134
sq2factor 217
sqrt 47
stack 214
standarddeviation 182
standardize 221
statevalf 222
statplots 222
stats 221
stellate 219
stepsize 87
Stirling! 207
stirling2 207
stopat 173
stoperror 173
stopwhen 173
string 21
structures 207
student 224
style 98
subgrel 210
submatrix 214
subs 44
subsets 207
subsop 44
substring 22
subvector 214
Sum 224
sum 24
sum2sqr 217
sumbasis 214
sumdata 182
sumrecursion 224

Sumtohyper 224
sumtohyper 224
sumtools 224
surfdata 219
swapcol 214
swaprow 214
Sylow 210
Sylvester 214
symmetrize 227
symmetry 120

tail 215
tangentcylindrical 134
tangentsphere 134
tassume 230
tau 217
tautology 214
taylor 217
tensor 225
tensorsGR 226
termscale 209
Testzero 157
tetrahedron 215
tetrahedron 215, 219
textplot 219
textplot3d 219
then 150
thickness 105
thue 217
time 160
tis 230
title 54
to 151
toeplitz 214
toroidal 134
torus 219
totorder 230
tpsform 220
trace 214
transform 219, 222, 227
translate 211, 219
transpose 214
triangle 209
Tripleint 224
TRUE 124
true 20
true 225
tubeplot 219
tuckmarks 129
tuttepoly 215
type 210

unapply 162
union 27
untrace 172

value 224
valuep 218
vandermonde 214
variance 182
vdegree 215
vecpotent 214
vectdim 214
vectoint 207
vector 214
verboseproc 175
vertices 215
view 120
void 215
vweight 215

wcollect 211
wdegree 208, 211
wedge 208
wedgeset 211
Weyl 226
whattype 22
While 151
with 205
writedata 180
wronskian 214
wsubs 211

xtickmarks 98

ytickmarks 98

zip 28

Двадцатый век некоторые называют атомным, другие — космическим, третьи — веком генетики. Мне кажется, что двадцатый век с не меньшим основанием можно назвать компьютерным. Современные вычислительные системы и базирующиеся на них информационные технологии совершенно изменили все стороны человеческого бытия. Пожалуй, в наибольшей степени изменился характер и повысилась производительность умственного труда. Теперь уже невозможно представить себе квалифицированного ученого, инженера, конструктора, не использующего Internet для получения и обмена самой свежей информации, программ для автоматизации выполнения и высококачественного оформления проектов. К числу наиболее замечательных программ такого типа можно отнести программу *Maple V* компании *Maple Waterloo*.

Программа достаточно легко осваивается, удобна в работе, так что ее может использовать даже школьник или студент для простых расчетов или для освоения математики. В то же время программа обладает настолько обширным набором функций и вычислительных средств, что она с успехом может быть применена для профессиональной работы в области математики и смежных дисциплин.

При постепенном освоении программы возникает ощущение, что ваши способности к точным наукам возрастают. На решение некоторых задач можно было бы истратить годы, в то время как с *Maple* вы справитесь с ней в считанные часы и даже минуты. За решение других задач вы бы вообще не взялись не будь у вас под рукой *Maple*.

1. Что такое Maple V

В самом общем смысле *Maple V* — это среда для выполнения математических расчетов на компьютере. В отличие от языков программирования высокого уровня, таких как *Фортран, БЕЙСИК, Си* или *Паскаль, Maple* может решать большое количество математических задач путем введения команд, без всякого предварительного программирования. Кроме того, *Maple* может оперировать не только приближенными числами, но и точными целыми и рациональными числами. Это позволяет получить ответ с высокой, в идеале с бесконечной, точностью.

Но, что самое важное, решение задач может быть получено аналитически, то есть в виде формул, состоящих из математических символов. Вследствие этого *Maple* называют также пакетом символьной математики.

Программа разработана исследовательской группой (*The Symbolic Computation Groop*) отделения вычислительной техники университета *Waterloo*, Канада, которая была образована в декабре 1980 Кейтом Геддом (Keith Geddes) и Гастоном Гонэ (Gaston Gonnet). Основное направление деятельности этой группы — исследования в области символьных вычислений (Symbolic Mathematical Computation), также называемой компьютерной алгеброй. Создание системы *Maple* — один из главных проектов группы.

Разработчики других известных математических пакетов, таких как *MathCad* и *MatLab* используют символьный процессор *Maple V* в своих программах. Кроме того, математические редакторы *Scientific WorkPlace* (на основе *Scientific Word*) и *MathOffice* (на основе *Microsoft Word*) для выполнения расчетов также дополнены символьным процессором *Maple V*.

К настоящему времени программа, благодаря усилиям разработчиков, превратилась в мощную вычислительную систему, предназначенную для выполнения сложных проектов. *Maple* умеет выполнять сложные алгебраические преобразования и упрощения над полем комплексных чисел, находить конечные и бесконечные суммы, произведения, пределы и интегралы, решать в символьном виде и численно алгебраические (в том числе трансцендентные) системы уравнений и неравенств, находить все корни многочленов, решать аналитически и численно системы обыкновенных дифференциальных уравнений и некоторые классы уравнений в частных производных. В *Maple* включены пакеты подпрограмм для решения задач линейной и тензорной алгебры, Евклидовой и аналитической геометрии, теории чисел, теории вероятностей и математической статистики, комбинаторики, теории групп, интегральных преобразований, численной аппроксимации и линейной оптимизации (симплекс-метод), а также задач финансовой математики и многих, многих других задач.

Maple V обладает также развитым языком программирования. Это дает возможность пользователю самостоятельно создавать команды и таким образом расширять возможности *Maple V* для решения специальных задач. Хороший текстовый редактор и прекрасные графические средства позволяют профессионально оформить выполненную работу.

В настоящем пособии будет описана последняя версия программы *Maple V 4.0 Power Edition*. Это полностью 32-разрядная программа, она почти одинаково, кроме некоторых незначительных отличий, работает в *Microsoft Windows*, на компьютере *Macintosh* или в системе *X — windows*. В настоящем пособии, если не оговорено иное, рассмотрена работа *Maple V* под управлением *Microsoft Windows 95*.

Необходимые требования к компьютеру:

Intel 386, 486, *Pentium* или полностью совместимый процессор;

от 18 до 42 МБ свободного дискового пространства;

минимум 8 МБ оперативной памяти;

Microsoft Windows 3.1x, *Windows NT 3.5* или *Windows 95*.

Если вы пользуетесь системой *Windows 3.1x*, то для поддержки 32-разрядного кода программы вам понадобится подсистема *Win32s*, подходящая версия которой также имеется в инсталляционном комплекте.

Книга не ставит своей целью охватить всю информацию по структуре и средствам *Maple*. Для этого существует документация пользователя и эквивалентные ей средства интерактивной помощи. Кроме того, поскольку программа имеет открытую архитектуру (большинство процедур написано на собственном языке *Maple*), могут быть прочитаны коды всех команд и функций. Цель настоящей книги — ознакомить читателя с возможностями программы *Maple V* и на конкретных примерах научить эффективно применять ее в своей работе.

2. Быстрый старт

Наиболее просто научиться работать с *Maple* и получать много полезных результатов можно в режиме командной строки, иначе называемом интерактивном режиме. При загрузке программы автоматически загружается новый рабочий лист (worksheet), на котором вы увидите приглашение для ввода команды > (prompt). В командную строку можно записать любое алгебраическое выражение, то есть выражение, состоящее из имен переменных и функций, чисел и символьных констант, соединенных алгебраическими операторами. Если в конце выражения поставить знак ";" (точка с запятой), то при нажатии клавиши **Enter** или кнопки с восклицательным знаком на инструментальной панели выражение будет обработано программой, а результат выведен на дисплей, например

```
> 2*3^5-x^2*sin(y-Pi);
```

$$486 + x^2 \sin(y)$$

Мы видим, что автоматически производятся арифметические действия и выводится результат.

Таким образом, мы можем получать вычисленные значения выражений, введенных в командную строку, то есть работать с программой, как с калькулятором. Мы можем также присваивать имена вводимым выражениям при помощи оператора присваивания :=, например

```
> R:=5/Pi*exp(x);
```

$$R := 5 \frac{e^x}{\pi}$$

Теперь можно ввести предыдущее выражение, просто записав присвоенное ему имя

```
> R;
```

$$5 \frac{e^x}{\pi}$$

Фактически каждое выражение, содержащее операторы и на конце которого стоит точка с запятой, является командой *Maple*, приводящей к выполнению операторов выражения. Однако в *Maple* используются и другого рода команды: команды-процедуры.

Такая команда вводится следующим способом:

```
> Имя_команды(аргумент, опции);
```

На конце команды обязательно должен стоять символ конца команды — точка с запятой или двоеточие. В противном случае команда не будет выполняться. Если поставлена точка с запятой, то команда будет выполнена и результат будет выведен на экран дисплея. Если после конца команды стоит двоеточие, то результат не будет выведен на дисплей, а только сохраниться в памяти компьютера.

Аргументом команды является в общем случае последовательность математических выражений, над которыми собственно говоря и выполняется команда.

Команды *Maple* очень короткие и простые, по названию легко понять их назначение. Легко также получить справку по любой команде, записав ее предположительное название после знака вопроса и нажав клавишу **Enter**.

Например, следующим образом можно получить справку для команды **expand**.

```
> ?expand
```

Следующие примеры иллюстрируют действие некоторых команд. Командой **combine** можно упростить тригонометрическое выражение:

```
> combine(sin(x)^4-cos(x)^4);
```

$$-\cos(2\,x)$$

Командой **plot3d** построить график поверхности (рис. 1)

```
> plot3d(sin(x*y),x=-Pi..Pi,y=-Pi..Pi);
```

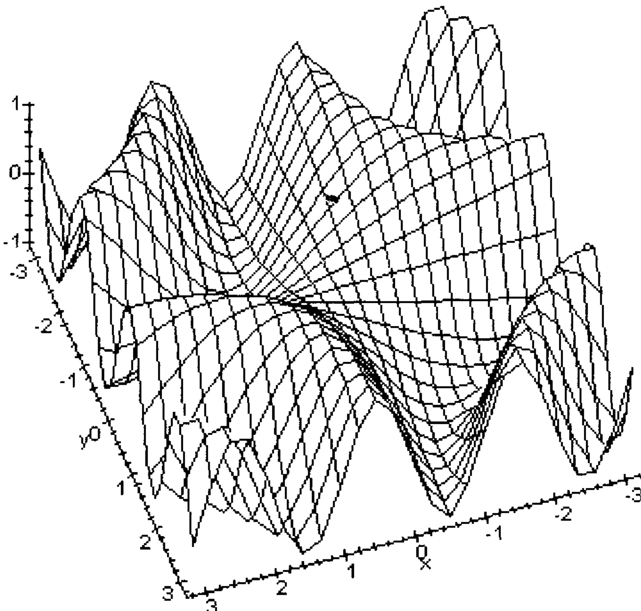

Рис. 1

Получив краткие сведения о *Maple* в этом разделе, вы можете уже начать экспериментировать с программой, решая некоторые математические задачи и черпая необходимые сведения о командах по мере надобности, однако для эффективной работы с программой необходимо ознакомиться с особенностями интерфейса, а также со структурой объектов, используемых в командах в качестве аргументов.

3. Интерфейс

Интерфейс пользователя поддерживает концепцию рабочих листов (“worksheets”), которые объединяют текст, входные команды, вывод и графику в одном документе (рис. 2).

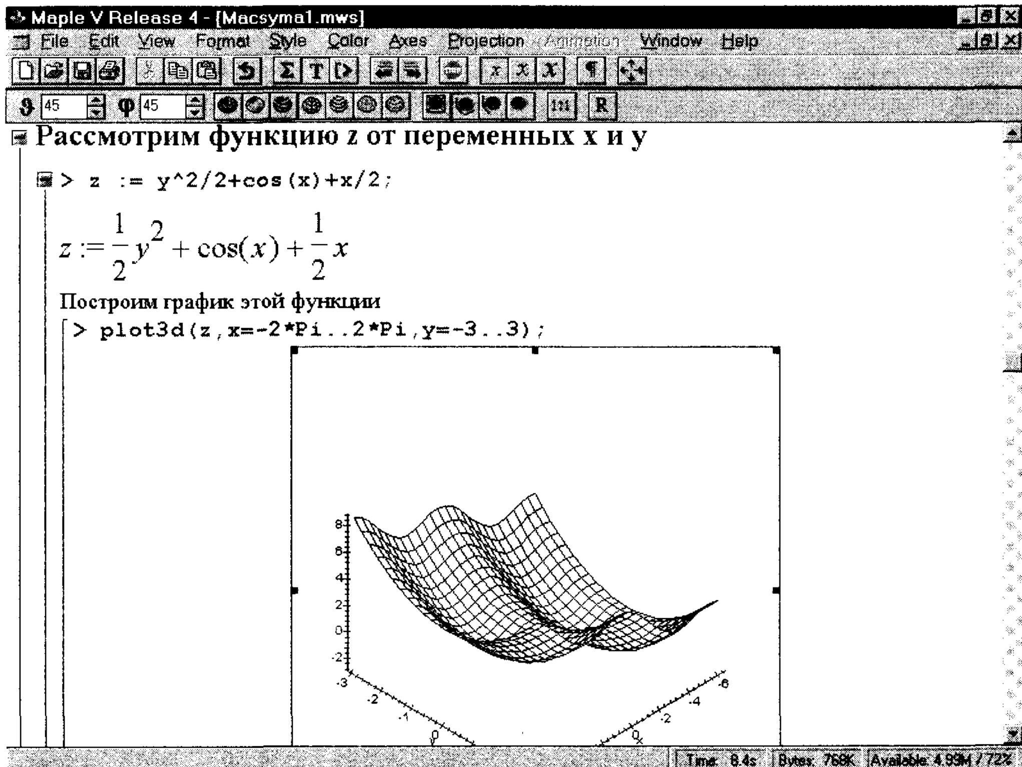

Рис. 2

Программа позволяет одновременно работать с несколькими рабочими листами и устанавливать между ними динамические связи, то есть переводить вычисления с одного листа на другой. Можно даже запускать несколько программ одновременно, что позволяет проводить сравнение вычислений при различных начальных значениях переменных.

В данном пособии ввод и вывод *Maple*, текст и графики изображены так, как они выглядят на рабочем листе *Maple*.

Запишем, например, следующую строку:

```
> plot(sin(x),x=0..Pi);
```

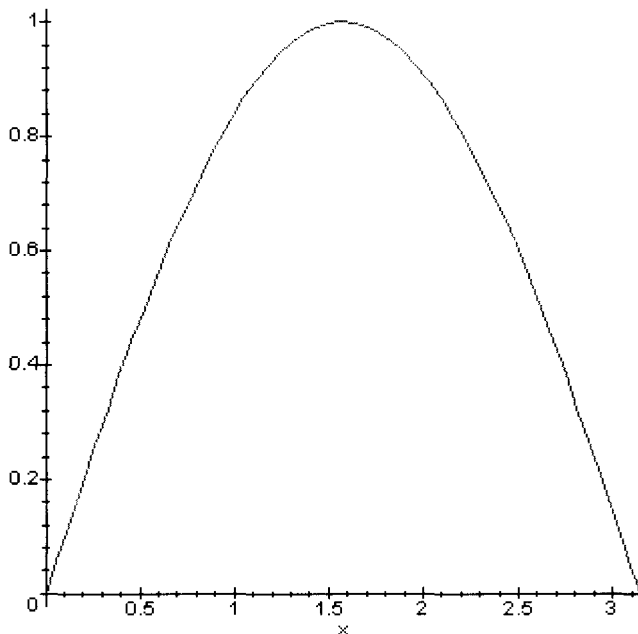

Рис. 3

Нажав на клавишу **Enter**, мы вызовем команду построения графика функции sin(x) с переменной x, меняющейся в интервале от 0 до Pi. Команда построит заданный график и выведет результат в поле вывода (рис. 3). Поле ввода и содержащее результат выполнения команды поле вывода команды на рабочем листе всегда охвачены общей скобкой слева. После выполнения команды вслед за полем вывода появляется новая командная строка с расположенным на ней курсором.

В зависимости от того, на каком объекте установлен курсор, изменяется вид строки меню и кнопок бара инструментов в соответствии с операциями, которые мы можем производить над объектом (рис. 2). Для того чтобы изучить назначение кнопок бара, достаточно включить опцию **balloone help** в разделе **Help** строки меню. После этого при фиксировании стрелки мыши на соответствующей кнопке или пункте меню будет высвечиваться сообщение в виде воздушного шара (balloon) с надписью.

Командную строку легко преобразовать в текстовую строку, нажав кнопку с изображением буквы **T** на панели инструментов или отметив пункт **Text input** в разделе **Insert** строки меню. Тогда с того места, где находится курсор, будет вводиться текстовая информация, которая не будет восприниматься командным процессором при вводе команды.

Текстовый редактор *Maple* позволяет форматировать отдельные знаки, слова, параграфы или целиком текст. В тексте (но не в командах) можно использовать все шрифты, инсталлированные в системе, в том числе кириллические, изменять их начертания (наклонный, полужирный и т. д.). При форматировании параграфов можно выбирать один из нескольких стилей или создать свой стиль.

Имеются заготовки для создания заголовков и подзаголовков четырех уровней. Есть возможность многоуровневой группировки командных полей и текстовых абзацев (рис. 2), причем можно сворачивать поля созданных групп для получения, например, списка заголовков. Можно создавать гипертекстовые связи, которые в тексте выделяются цветом (исходно — зеленым цветом). Редактор содержит также функцию поиска по одному или нескольким введенным символам.

Качество вывода (правильность отображения математических символов) также можно изменять выбором соответствующей опции раздела **Options** строки меню. При этом если мы выберем **Typeset notation** (полиграфическое изображение), то качество изображения выводимых математических формул действительно будет очень высоким. Кроме того, такое полиграфическое качество можно обеспечить также для вводимых в тексте (командной строке) формул. Для этого просто нужно нажать на инструментальной панели кнопку с изображением Σ (отжать кнопку **x**) или выделить опцию **Maple input** пункта строки меню **Insert**.

При этом справа от кнопки ввода команды, изображаемой восклицательным знаком, появится строка ввода формул (рис. 4). В этой строке вводимые символы будут выглядеть так, как в командной строке, однако в текстовой строке они будут преобразованы в полиграфический формат. В качестве примера на рис. 4 показано, как выглядят на рабочем листе в полиграфическом формате командная строка и формула в тексте.

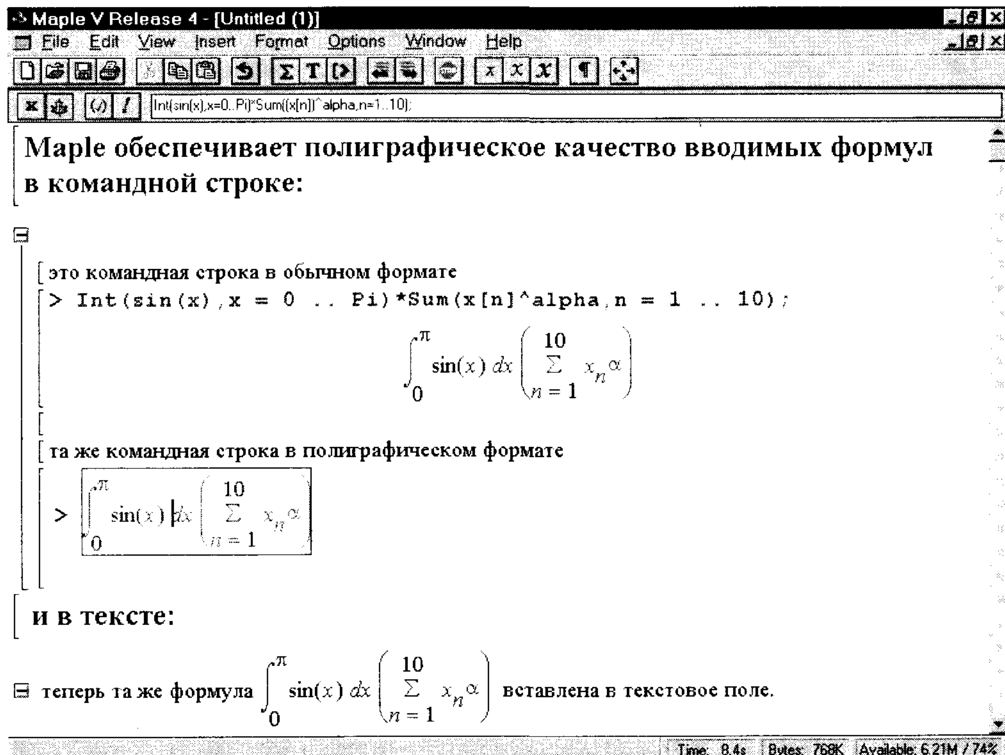

Рис. 4

Остановимся еще на средствах помощи *Maple*. Программа содержит полную информацию по всем командам, функциям и интерфейсу пользователя, а также статьи обучающего характера для начинающего пользователя. Статьи снабжены примерами, которые можно использовать как заготовки для ваших выкладок. Как уже упоминалось выше, получить информацию по конкретной команде или функции *Maple* можно, введя ее наименование с предшествующим знаком вопроса в командной строке. Однако это не единственный метод поиска необходимой информации. Из раздела **Help** строки меню можно вызвать пункт **Contents** (содержание), при этом вы можете просмотреть названия всех статей справки аналогично тому, как вы просматриваете содержание глав книги. Можно также осуществить тематический поиск, введя из диалогового окна пункта **Topic search** меню название темы для поиска. Кроме того, есть возможность поиска необходимой информации по всему тексту статей справки. Для этого нужно воспользоваться пунктом меню **Full text search**. В диалоговом окне введите необходимые ключевые слова для поиска и вы получите список статей, в которых содержится необходимая информация. Вы сможете получить доступ к поиску по всему тексту, если выделите необходимое слово на рабочем листе и нажмете клавишу F1.

4. Объекты Maple

4.1. Язык программы

Определение языка можно разбить на четыре части: символы (characters), высказывания (tokens), синтаксис (syntax) и семантика (semantics) — толкование.

Элементы языка

Набор символов включает 26 прописных букв латинского алфавита, 26 строчных букв, 10 цифр и еще 32 специальных символа.

Высказываниями (лексемами) являются ключевые слова, операторы программирования, строки, натуральные числа и знаки препинания.

Зарезервированные слова

Они имеют специальное значение и их нельзя применять в качестве переменных в программах.

Не только элементы структуры языка, но и наименования функций, команды *Maple*, наименования типов объектов могут иметь зарезервированное значение, но только первые нельзя использовать в качестве переменных, остальные можно — в некоторых контекстах.

Операторы программного языка

Имеется три типа операторов (binary, unary, nullary): двуместные (бинарные), одноместные (унарные) и нульарные — не имеющие операндов. Последних всего три (**ditto**-операторы) обращения к предыдущему вычислению (","",""").

Разделители высказываний

Можно разделять высказывания пустыми разделителями или знаками препинания.

Пустые разделители

Это пробелы, знаки табуляции и возврат каретки. Пробелы нельзя использовать внутри высказываний (лексем).

```
> a:=b;
                                  a := b

> a: =b;v

Syntax error, `=` unexpected
```

В то же время пробелы можно использовать между лексемами. В строках, охваченных обратными кавычками, они становятся частью высказывания.

Все символы строки за решеткой # *Maple* интерпретирует как часть комментария.

```
> a * x + x*y; # Это комментарий
```

$$a\,x + x\,y$$

Перейти на новую строку с продолжением записи команды можно через **Shift-Enter**.

4.2. Структура объектов

Выражения

Как уже упоминалось, объекты, с которыми оперирует *Maple,* являются математическими выражениями. Наиболее простые выражения состоят из одного числа или переменной. В общем случае выражения *Maple* могут состоять из тысяч и тысяч чисел и имен, соединенных при помощи арифметических операторов. Арифметические операторы *Maple* включают:

+ сложение
− вычитание
* умножение
/ деление
^ возведение в степень

Далее — некоторые примеры простых выражений.

```
> a+b+c;
```

$$a + b + c$$

```
> 3*x^3-4*x^2+x-7;
```

$$3\,x^3 - 4\,x^2 + x - 7$$

```
> x^2/25+y^2/36;
```

$$\frac{1}{25\,x^2} + \frac{1}{36\,y^2}$$

Порядок выполнения операций в выражениях соответствует стандартной форме старшинства операторов, применяемой в математике. Если возможны любые неоднозначности, используйте круглые скобки ( ), чтобы определять порядок операторов.

```
> 2+3 * 4-5;
```

$$9$$

```
> (2+3) * 4 -5;
```

$$15$$

```
> (2+3) * (4-5);
```

$$-5$$

```
> (a+b)/(a*c);
```

$$\frac{a+b}{a\,c}$$

Если набор скобок избыточен, синтаксический анализатор будет устранять их в процессе вычислений.

Числа и константы, строки и имена

Числа, строки и имена — самые простые объекты в *Maple* и в то же время — самые простые выражения.

1. Целые и рациональные числа

Так как *Maple* — программа, оперирующая с символами, числа не всегда выражаются в десятичном представлении. Целые числа выражаются просто цифрами в десятичной записи, рациональные числа используют оператор деления, чтобы выделить числитель и знаменатель:

```
> 25;
```

$$25$$

```
> 28/70;
```

$$\frac{2}{5}$$

Как видите, рациональные числа автоматически упрощаются. При необходимости используется десятичное представление точных значений рациональных чисел, которое также может явиться результатом многих вычислений программы. Эти числа могут быть записаны также в научном представлении (с использованием степени 10).

Примеры чисел с плавающей точкой

```
> 2.3;
```

$$2.3$$

```
> .143*10^(-44);
```

$$.1430000000\ 10^{-44}$$

```
> Float(3141, -3);
```

$$3.141$$

2. Математические константы

Кроме констант, которые можно вводить в виде целых, рациональных или с плавающей точкой чисел, *Maple* содержит также большое количество общеизвестных математических констант.

Среди них:

Pi — 3.1415928535...
exp (1) — основание натурального логарифма
I — квадратный корень из −1
infinity — (бесконечность)
−infinity — (минус бесконечность)
gamma — константа Эйлера
Catalan — константа Каталана
true, false — (истина, ложь) — булевы константы

Необходимо следить за правописанием при использовании этих констант (включая строчные и прописные буквы), например Pi и pi не эквивалентны.

3. Смешивание и совместимость различных типов констант

Как обсуждалось прежде, возможность выражать числовые значения в их точном представлении (например 1/3 а не .3333...) — одно из преимуществ символьной алгебры. Обычно значения в их точной форме могут сохраняться в течение вычислений. Однако в ряде случаев точные значения будут преобразованы в приближенные. Один из таких случаев — когда смешиваются типы в выражениях.

Следующие примеры иллюстрируют вышесказанное:

```
> 1/3+2;
```

$$7/3$$

```
> 1/3+2.0;
```

$$2.333333333$$

```
> Pi/6.;
```

$$.1666666667\ Pi$$

4. Строки

Строка в *Maple* состоит из некоторого количества любых символов, заключенных в обратные кавычки (backquote, “ ` ”), изображение этой кавычки соответствует знаку апострофа.

Далее — некоторые примеры строк.

```
> `Это — Maple строка`;
```

Это — Maple строка

```
> `12+3abc`;
```

12 + 3abc

```
> `inv/ert.src`;
```

inv/ert.src

Как видите, специальные символы (+,., /, и т.д.) могут включаться в любом месте строки, если имеются кавычки. Если кавычки отсутствуют, тогда эти специальные символы интерпретируются как обычные операторы:

```
> 3+abc+4;
```

$7 + abc$

```
> directory/filename;
```

$$\frac{directory}{filename}$$

```
> invert.src;
```

invertsrc

5. Имена

Выражения *Maple* могут присваиваться именам. Имя состоит из специального типа строки, которая в самой простой форме является буквенным символом (a—z, A—Z) и может сопровождаться одним или большим количеством буквенных символов, цифрами (0—9), и символом подчеркивания (_). Имя может достигать длины вплоть до 524 275 знаков. Имена чувствительны к регистру, то есть имя Name отличается от имени name. Кроме этого именем может служить любая строка, то есть любой набор символов, заключенных в обратные кавычки. Имя, обозначенное строкой, состоящей из разрешенных для имени символов совпадает с именем из тех же символов без кавычек. Например `Name5` и Name5 — одно и то же имя. Имена, начинающиеся символом подчеркивания, используются в *Maple* как имена глобальных переменных.

Две обратные кавычки, вводимые последовательно в начале строки, интерпретируются как одна кавычка. Это позволяет включать символ обратной кавычки в текст строки.

Функция **type** различает имена двух типов: **string** (строка) и **indexed** (индексное).

Приведем примеры:

```
> `обратные `` кавычки`;
```

обратные ` кавычки

```
> length(`Very long string`);
```

```
> substring(` abcdefghijklmnopqrstuvwxyz `, 15..20);
```

nopqrs

Приведем некоторые примеры допустимых имен.
Строковые имена:

```
> MyVariable;whattype(");
```

MyVariable

string

```
> hello;
```

hello

```
> `greatest variable`;
```

greatest variable

Индексные имена:

```
> A[1];whattype(");
```

A_1

indexed

```
> A[i,j];
```

$A_{i,j}$

```
> A[i][j];
```

$A_{i\,j}$

Примеры недопустимых в *Maple* имен.

```
> the+quotient;
```

the + quotient

```
> ... etc;
Syntax error, `..` unexpected
```

```
> 45opt;
Syntax error, missing operator or `;`
```

6. Оператор конкатенации

Для объединения строк применяется оператор конкатенации, который записывается в виде

cat (\`a\`, \`b\`, \`c\`, ...),

где \`a\`, \`b\`, \`c\`, ... — строки. Результатом оператора является строка \`abc...\`.

Удобный инструмент в конструировании строк и имен является также знак конкатенации (.). При помощи этого знака можно создавать нумерованные наборы имен. Однако при использовании символа точки для конкатенации соблюдайте осторожность, чтобы не возникла путаница между точкой в десятичном представлении числа и точкой — знаком конкатенации. Запомните правило: имя должно всегда находиться слева от знака конкатенации.

Некоторые примеры использования.

```
> seq(name.i,i=1..5);
```

name1, name2, name3, name4, name5

```
> add(A.i,i=1..5);
```

$$A1 + A2 + A3 + A4 + A5$$

7. Использование кавычек в Maple

Помимо обратных кавычек, используемых для создания строк, имеются еще два типа кавычек. Очень важно понимать, как использовать те или иные кавычки, и не путать их между собой.

Наиболее просто запомнить назначение двойных кавычек (ditto оператор). Двойные кавычки (") вызывают предыдущий вывод в сеансе *Maple*. Один набор двойных кавычек вызывает результат предыдущей команды, два набора ("") — результат команды, выполненной до предыдущей, и три набора (""") вызывают еще более ранний результат. Однако четыре набора кавычек уже не применяются. Использование двойных кавычек в сложных программах, состоящих из нескольких команд, может привести к непредсказуемым результатам. Правильно использовать двойные кавычки в режиме командной строки. Даже при использовании в качестве признака конца команды двоеточия (:), чтобы подавить вывод на дисплей, для последующего вызова результата можно применить оператор двойных кавычек. Другими словами, оператор двойных кавычек играет роль краткосрочной замены оператора присваивания для сокращения объема вводимой информации. В общем случае, чтобы получить возможность обращения к результату некоторой команды впоследствии, этому результату присваивают имя.

Возможно, наиболее трудно понять смысл использования прямой кавычки (ее изображение похоже на знак ударения). В упрощенном изложении оператор, заключенный в прямые кавычки, освобождается от них при однократном вводе, то есть происходит задержка выполнения этого оператора на один проход через синтаксический анализатор *Maple*. Иначе говоря, каждый раз, когда

синтаксический анализатор сталкивается с выражением, заключенным в прямые кавычки, он удаляет внешний слой этих кавычек. При двойном обрамлении выражения прямыми кавычками выполнение оператора задержится на два прохода и так далее.

```
> "(x^2-x-2)";
```

$$'\mathrm{factor}(x^2 - x - 2)'$$

```
> 'factor(x^2-x-2)';
```

$$\mathrm{factor}(x^2 - x - 2)$$

```
> factor(x^2-x-2);
```

$$(x + 1)\,(x - 2)$$

Укажем два наиболее частых случая использования прямых кавычек. Во-первых, прямые кавычки могут использоваться для отмены присваивания какого-либо значения некоторой переменной. Присваивание переменной x некоторого значения записывается при помощи оператора присваивания :=.

Пусть, например

```
> x:= 3;
```

$$x := 3$$

Проверим теперь значение x, введя просто

```
> x;
```

$$3$$

Мы видим, что переменной x присвоено значение 3. Чтобы отменить это присваивание, запишем

```
> x:= 'x';
```

$$x := x$$

Теперь, введя x, получим

```
> x;
```

$$x$$

Во-вторых, прямые кавычки используются внутри команд с индексными параметрами (**sum**, **product**), например

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

Последовательности выражений

Выражения не самый сложный объект в *Maple*. Один из более сложных объектов — последовательность выражений. Последовательность выражений — просто несколько выражений, отделенных запятыми. Большинство команд требуют ввода последовательности выражений в виде параметров, и многие из них возвращают результат, который также включает последовательность выражений. Самый простой способ создавать последовательность выражений — просто ввести ее следующим образом.

```
> 1,2,3,4,5;
```

$$1, 2, 3, 4, 5$$

```
> a+b, b+c, c+d, e+f, f+g;
```

$$a+b, b+c, c+d, e+f, f+g$$

В качестве альтернативы имеются еще два способа создавать неявную последовательность выражений.

Во-первых, с этой целью может использоваться оператор **$** (один либо совместно с оператором диапазона, записываемым в виде многоточия "..". Этот оператор создает упорядоченные последовательности.

Приведем примеры:

```
> a$6;
```

$$a, a, a, a, a, a$$

```
> $1..6;
```

$$1, 2, 3, 4, 5, 6$$

```
> i^2$i=1..6;
```

$$1, 4, 9, 16, 25, 36$$

```
> i:=evaln (i);
```

$$i := i$$

```
> 2 * i$i=1..10;
```

$$2, 4, 6, 8, 10, 12, 14, 16, 18, 20$$

```
> a[i] $ i = 1..3;
```

$$a_1, a_2, a_3$$

Во-вторых, имеется команда **seq**, которая работает следующим образом:

```
> seq ( i!/i^2, i=1 ..7 );
```

$$1, \frac{1}{2}, \frac{2}{3}, \frac{3}{2}, \frac{24}{5}, 20, \frac{720}{7}$$

```
> seq(D(f), f=[sin,cos,tan,exp,ln]);
```

$$\cos, -\sin, 1 + \tan^2, \exp, a \to 1/a$$

Преимущество команды **seq** в том, что она очень быстрая и может использоваться в ряде ситуаций для увеличения скорости вычисления.

В следующем примере показано, как последовательность выражений используется в качестве аргумента в команде **max**(_).

```
> max(Pi,exp(1),tan(5*Pi/6));
```

$$\pi$$

Наборы и списки

1. Наборы

Набор — неупорядоченная совокупность выражений. Любое допустимое выражение может содержаться в наборе. Наборы часто используются как ввод в процедуре *Maple* и часто содержатся в выводе. Набор записывается как последовательность выражений, заключенная в фигурные скобки {}. Необходимо сделать одно важное замечание, касающееся наборов — повторные элементы автоматически удаляются из набора. Эта особенность очень удобна для программирования большого количества задач. Первый из следующих трех примеров демонстрирует это правило.

```
> { 1, 1, 2, 3, 2 };
```

$$\{1, 2, 3\}$$

```
> {a*x, my.name, -234.456, `Учебное пособие
  по Maple!`};
```

{*myname*, *a x*, −234.456, *Учебное пособие по Maple!*}

```
> {`blue`, `red`, `white`};
```

{*red*, *blue*, *white*}

Как видно из примеров, порядок, в котором записаны элементы набора, не обязательно совпадает с порядком, в котором их воспринимает *Maple*.

2. Оперирование элементами набора (команды union, intersect, minus)

По сути, наборы можно рассматривать как множества объектов *Maple*. Так же как в теории множеств, для оперирования с наборами в *Maple* введены три основных оператора: оператор объединения (**union**) объединяет элементы двух наборов в один (исключая при этом любые повторные элементы); оператор пересечения (**intersect**) создает набор, который содержит любые элементы, общие двум начальным наборам; и оператор исключения (**minus**) — удаляет из первого набора любые элементы, содержащиеся во втором наборе.

```
> { a, b, c, d } union { d, e, f};
```

$$\{b, c, f, e, d, a\}$$

```
> {1, 2, 3, 4, 5} intersect {2, 4, 6, 8, 10};
```

$$\{2, 4\}$$

```
> {x1, x2, x3} minus {x1, y1};
```

$$\{x2, x3\}$$

3. Списки

Списки, так же как наборы, определяются последовательностями выражений, однако списки заключаются в квадратные скобки "[]". Будучи близкими по написанию, списки и наборы существенно различаются. Списки — "хорошо упорядоченные объекты". Это означает, что порядок, в котором записан список, будет точно также восприниматься *Maple* и будет сохраняться в течение вычислений. Другое важное отличие — повторяющиеся элементы не удаляются из списка.

Далее — некоторые примеры списков:

```
> [1, 2, 3, 4, 5, 4, 3, 2, 1];[a, d, c, b, e ];
```

[1, 2, 3, 4, 5, 4, 3, 2, 1]

[a, d, c, b, e]

```
> [ { c, a, t }, { d, o, g }, { m, o, u, s, e } ];
```

[{c, t, a}, {g, d, o}, {e, u, s, m, o}]

В последнем примере каждый из трех наборов — элемент списка. В то время как порядок элементов внутри наборов может изменяться, порядок самих трех наборов остается неизменным. Хотя операторы объединения, пересечения и исключения не воздействуют на списки, для извлечения и манипулирования элементами списка могут использоваться команды **op** и **nops** (смотрите далее).

4. Оперирование элементами списка (команды select, remove, zip, sort)

Для извлечения элементов из списка по некоторому признаку существует команда **select**, которая по заданному правилу (логическому соотношению, являющемуся первым параметром аргумента команды) выбирает из списка (второй параметр) элементы и выводит их в той же последовательности в список.

```
> large:=x-> is(x>3);
```

$$large := x \to \text{is}(3 < x)$$

```
> L:=[8,2,95,Pi,sin(9)];
```

$$L := [8, 2, 95, \pi, \sin(9)]$$

```
> select(large,L);
```

$$[8, 95, \pi]$$

Команда **remove**, наоборот, выбрасывает из списка удовлетворяющие заданному правилу элементы и выводит список оставшихся элементов.

```
> remove(large,L);
```

$$[2, \sin(9)]$$

Возможно также извлечь элементы определенного типа командой **type**:

```
> select(type,L,numeric);
```

$$[8, 2, 95]$$

Объединение двух списков

Определим два списка X и Y

```
> X:=[seq(ithprime(i),i=1..6)];
```

$$X := [2, 3, 5, 7, 11, 13]$$

```
> Y:=[seq(binomial(6,i),i=1..6)];
```

$$Y := [6, 15, 20, 15, 6, 1]$$

Можно эти два списка объединить в один при помощи команды

```
> [op(X),op(Y)];
```

$$[2, 3, 5, 7, 11, 13, 6, 15, 20, 15, 6, 1]$$

Для объединения списков по заданному правилу применяется команда **zip**. Первым параметром команды **zip** задается правило объединения пар элементов. Можно, например, элементы двух списков объединить попарно в список наборов (по одному элементу из каждого списка в наборе).

```
> zip((x,y)->{x,y},X,Y);
```

$$[\{2, 6\}, \{3, 15\}, \{5, 20\}, \{7, 15\}, \{6, 11\}, \{1, 13\}]$$

Это правило объединяет элементы списков попарно в список списков

```
> pare:=(x,y)->[x,y];
```

$$pare := (x, y) \to [x, y]$$

```
> P:=zip(pare,X,Y);
```

$$P := [[2, 6], [3, 15], [5, 20], [7, 15], [11, 6], [13, 1]]$$

Такой список пар списков применяется для построения графика по заданным точкам плоскости (рис. 5).

```
> plot(P);
```

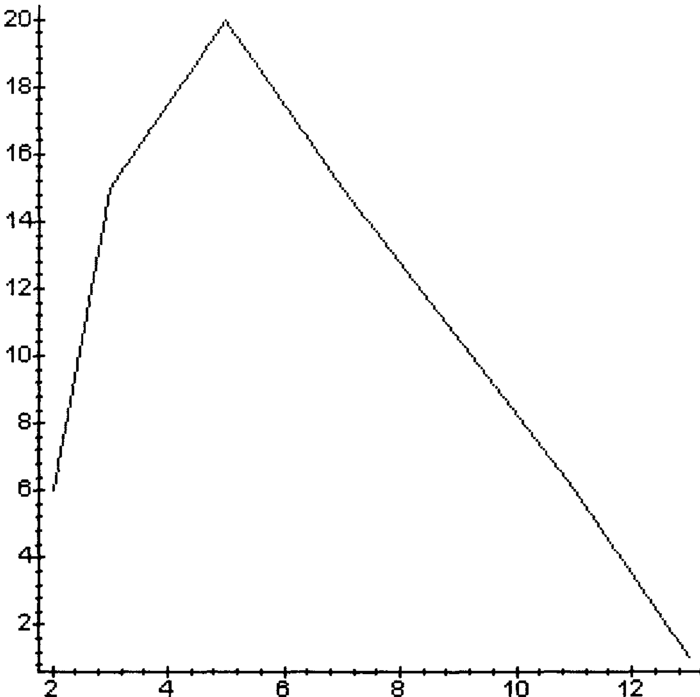

Рис. 5

Еще два примера иллюстрируют применение знака конкатенации

```
> zip((x,y)->x.y,[a,b,c,d,e,f],[1,2,3]);
```

$$[a1, b2, c3]$$

```
> zip((x,y)->x.y,[a,b,c,d,e,f],[1,2,3],5);
```

$$[a1, b2, c3, d5, e5, f5]$$

Сортировка списка по заданному правилу. В этой команде список является первым параметром аргумента, а правило — вторым.

```
> sort([3.12,1,1/2],(x,y)->evalb(x>y));
```

$$[3.12, 1, 1/2]$$

```
> bf:=(x,y)->is(x<y);
```

$$bf := (x, y) \to \mathrm{is}(x < y)$$

```
> sort([4.3,Pi,2/3,sin(5)],bf);
```

$$[\sin(5), 2/3, \pi, 4.3]$$

Операторы присваивания и уравнения

Мы уже встречались ранее с оператором присваивания ":=", который используется для присваивания значения некоторому имени. Этот раздел объясняет различие между оператором присваивания, обозначаемым ":=" (символ двоеточия, сопровождаемый знаком равенства =), и оператором уравнения, обозначаемым знаком равенства =.

Следует сделать еще несколько замечаний относительно оператора присваивания. При использовании оператора присваивания *Maple* помнит только последнее присвоенное значение для любой переменной. Если вы присвоите переменной x значение 5, а потом — значение 75, то запомнится только последнее присваивание. Вы можете переопределить любое использованное вами имя для команды или другого объекта, однако *Maple* не позволит вам использовать имя для переменной, если оно используется в качестве имени одной из встроенных команд, функций или констант *Maple*. Имена этих объектов защищены и вы получите сообщение об ошибке. Кроме того, можно, используя команду **protect**(имя) защитить любое введенное вами имя.

Оператор уравнения (знак равенства "="), в отличие от рассмотренного выше оператора присваивания, просто связывает между собой некоторые переменные и значения выражения. Уравнения не присваивают явных значений переменным, которые они содержат.

Например:

```
> x = y + 3;
```

$$x = y + 3$$

```
> x;
```

$$x$$

```
> y;
```

$$y$$

Как видите, переменным x и y ничего не присваивается. Оператор "=" чаще всего употребляется или в параметре команды *Maple* или в выводе результата. Очень полезное семейство команд, использующих оператор "=", — команды решения уравнений различного вида:

- **solve** предназначена для аналитического решения линейных и нелинейных уравнений, неравенств и систем;
- **fsolve** предназначена для численного решения линейных и нелинейных уравнений, неравенств и систем;
- **dsolve** решает набор обыкновенных дифференциальных уравнений;
- **rsolve** решает набор рекуррентных уравнений.

Приведем примеры.

```
> sols:= solve({x+y=3, x-y=1}, {x, y});
```

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

```
> x;
```

$$x$$

```
> y;
```

$$y$$

Полученное решение — набор уравнений для определения переменных. Если имеется много решений, они все будут получены. В то же время переменным x и y в вышеупомянутом примере значения решений не присваиваются. Для присваивания решений исходным переменным нужно использовать команду **assign**, которая в уравнении (или наборе уравнений) заменяет каждый оператор "=" на оператор ":=".

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

$$2$$

```
> y;
```

$$1$$

```
> x:='x';
```

$$x := x$$

```
> y:='y';
```

$$y := y$$

Другое частое использование знака равенства — в операторах булевых (логических) выражений. Когда необходимо выяснить характер зависимости между значениями двух переменных, оператор "=" может использоваться для проверки равенства. Другие булевы операторы сравнения — : <, <=, <>, and, or, not. Команда **evalb** проверяет, является ли булево соотношение истинным или ложным.

Приведем примеры.

```
> evalb ( 3! = 4!/2^2 );
```

$$true$$

```
> evalb ( 157/50 > 22/7 );
```

$$false$$

```
> evalb ( isprime (5) and isprime (541) );
```

$$true$$

Функции

Maple имеет несколько способов представления функции. Во-первых, если мы какому-либо выражению присвоим имя, то фактически присвоенное имя является функцией переменных, стоящих в выражении. Например

```
> p:=x^2+2*x+1:
> p;
```

$$x^2 + 2\,x + 1$$

При помощи оператора присваивания := в строке 2 мы присвоили переменной p значение многочлена $x^2 + 2\,x + 1$. Теперь, просто введя присвоенное имя p в строке 3, мы получили значение этого многочлена. В то же время переменная x осталась незаданной, что легко проверить вводом

```
> x;
```

$$x$$

Теперь, если мы введем

```
> x:=2;
```

$$x := 2$$

то получим при вводе p число 9, то есть значение многочлена при x=2. Таким образом значение переменной p определяется значением математического выражения, которое присвоено переменной p. Присваивая переменной x разные значения, мы будем получать вычисленные по формуле значения p. Таким образом, переменная p фактически является функцией x.

Мы можем применить эту переменную в правой части другого оператора присваивания, например

```
> q:=p^3+1;
```

$$q := 730$$

В этом отличие языка *Maple* от обычных языков программирования — в качестве переменных в математических выражениях могут использоваться запрограммированные имена. Однако при записи операторов присваивания следует соблюдать осторожность. Если в левой и правой частях таких операторов будут стоять одинаковые переменные, которым еще ничего не присвоено, например

```
> C:=C^2+1;
Warning, recursive definition of name
```

$$C := C^2 + 1$$

то программа выдаст предупреждение, так как вычисление такого присваивания приведет к бесконечному циклу. Точно также нельзя, например, записать команду вычисления неопределенного интеграла

```
> int(p^2,p);
Error, (in int) wrong number (or type) of arguments
```

так как переменная p уже не является независимой — ей присвоено значение $x^2 + 2\,x + 1$.

В *Maple* можно отменить присваивание такими командами:

```
> p:=evaln(p);
```

$$p := p$$

или

```
> p:='p';
```

$$p := p$$

тогда ввод

```
> int(p^2,p);
```

$$\frac{1}{3}\,p^3$$

выполняет интегрирование функции p^2.

Мы видим, что при задании функции методом присваивания имени выражению имеются некоторые неудобства ее использования.

Чтобы не снимать при каждом вызове функции с переменных численные присваивания, можно использовать команду замены. Пусть, например

```
> F:=x^3*sin(t);
```

$$F := x^3 \sin(t)$$

```
> subs({x=3, t=Pi/2},F);
```

$$27 \sin(\frac{1}{2}\pi)$$

Существует и еще одно неудобство — определенную таким образом функцию невозможно использовать для расширения библиотеки команд *Maple*.

Более общий и наиболее употребительный метод задания функции — путем определения процедуры. С общим определением процедур *Maple* мы познакомимся в разделе, посвященном программированию. Сейчас мы рассмотрим специальный вид процедур — функциональные операторы.

Функциональный оператор задает функцию или последовательность функций от одной или нескольких переменных. Он записывается в виде

(последовательность переменных) -> (последовательность выражений), например

```
> F1:=(x,t) -> (x^3 + sin(t), exp(x)-ln(x+1));
> F2:=(x,t) -> x^2+t^2;
> F3:=x -> (sin^(x+1)*x,cos^(x-1)/x);
```

$$F1 := (x, t) \rightarrow (x^3 + \sin(t), e^x - \ln(x+1))$$

$$F2 := (x, t) \rightarrow x^2 + t^2$$

$$F3 := x \rightarrow \left(\sin^{(x+1)} x, \frac{\cos^{(x-1)}}{x}\right)$$

Чтобы получить значение функции при некоторых значениях переменных, достаточно записать их в качестве параметров в той же последовательности, в которой они указаны в команде, например

```
> F1(y,tau);
```

$$y^3 + \sin(\tau)\, e^y - \ln(y+1)$$

Другой способ задания функционального оператора — использование команды **unapply**.

Эта команда преобразует любое математическое выражение в функцию от указанных в команде переменных, содержащихся в этом выражении, например

```
> unapply(x^3 + sin(t),x,t);
```

$$(x, t) \text{ -> } x^3 + \sin(t)$$

```
> unapply([sin^(x+1)*x,cos^(x-1)/x],x,t);
```

$$(x,t) \rightarrow \left[\sin^{(x+1)} x, \frac{\cos^{(x-1)}}{x}\right]$$

Операторы Maple

Помимо упомянутых выше арифметических операторов, логических операторов, **ditto**-оператора, функционального оператора *Maple* содержит большое количество других операторов. Операторы играют большую роль в формировании выражений и выполнении математических расчетов. Подробнее об операторах и программировании их свойств смотрите в разделе 8.2. Здесь мы опишем два часто используемых оператора Maple.

1. Оператор композиции @

Этот оператор применяется для создания сложной функции. Он записывается в виде

- f@g — для создания композиции функций f и g или
- f@@n — для n-кратного применения функции f, например

```
> (ln@sin)(x);
```

$$\ln(\sin(x))$$

```
> f:= x-> 1/(1+x);(f@@5)(x);
```

$$f := x \rightarrow \frac{1}{1+x}$$

$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{1+x}}}}}$$

На следующем примере при помощи таблицы обратных функций **invfunc**, загруженной из библиотеки, мы создали функцию g, обратную функции f.

```
> readlib(invfunc):invfunc[f]:=g;
```

$$invfunc_f := g$$

Так можно упростить композицию взаимно обратных функций.

```
> simplify(f@g@@2);
```

$$g$$

2. Нейтральный оператор

Нейтральные операторы определяются пользователем. Имя нейтрального оператора должно начинаться с символа & и может сопровождаться допустимым *Maple*-именем и некоторыми специальными символами кроме &, |, (), [], {}, ;, :, ` `, #, "пробел".

Свойства определяемого пользователем оператора задаются при помощи команд

- **define**(aa(oper)), где oper — имя определяемого оператора, aa — имя абстрактного алгебраического объекта или (в следующем примере — линейный) или
- **define**(oper, property1, property2, ...), где oper — имя определяемого оператора, property — свойство оператора.

Эти команды определяют правила вычисления и упрощения оператора.

Приведем пример

```
> define(Linear(`&L`));
> &L(5*x+3*y);
```

$$5\ \&L(x) + 3\ \&L(y)$$

4.3. Определение типов объектов

Каждому выражению (и другому объекту) в *Maple* соответствует связанный с ним тип объекта. Базисными типами объектов для выражений являются: **string**, **integer**, **fraction**, **float**, арифметические операторы `+`, `*`, `^`, и **function**. Для определения типа объекта используется команда **whattype**:

```
> whattype (15/37);
```

fraction

```
> whattype ([1,2,3,4,5]);
```

list

```
> whattype (( x+3 ) * (y-4));
```

*

Хотя всегда можно запросить *Maple* о типе объекта, заранее задать тип объекта невозможно. Нельзя задать как, например, в Фортране, чтобы переменная j была всегда целой. При программировании с *Maple* имеются случаи, когда необходимо производить различные вычисления в зависимости от типа переменной. Команда **type** позволяет сделать запрос типа переменной.

```
> greetings := `best regards`;
```

greetings := best regards

```
> type (greetings, integer);
```

false

```
> type (greetings, string);
```

true

```
> whattype (x = y+1);
```

=

Имеются еще две полезные команды для анализа структуры объектов — команда **hastype**, которая сообщает, содержит ли объект подобъект данного типа, и команда **has**, которая сообщает, содержится ли определенный подобъект в объекте.

Приведем примеры:

```
> hastype ((x+1/2) * exp (3), fraction);
```

true

```
> hastype (x^2+3*x+5, `*`);
```

true

```
> has(x^2+3 * x+5, 3);
```

true

```
> has(x^2+3 * x+5, 2 * x);
```

false

```
> hastype (int (exp (-x^2), x), fraction);
  int (exp (-x^2), x );
```

$$true$$

$$\frac{1}{2}\sqrt{\pi}\,\mathrm{erf}(x)$$

В то время как эти примеры довольно очевидны, **hastype** и **has** неоценимы при работе с очень большими объектами.

4.4. Анализ структуры объектов

Каждый объект *Maple* состоит из подобъектов известного типа, которые также состоят из меньших подобъектов, и так вплоть до элементарных базисных объектов. Наглядно можно представить каждый объект в виде древовидной структуры. Средства *Maple* позволяют исследовать и извлекать индивидуальные элементы, составляющие объект. Эта возможность очень полезна при оперировании большими объектами. В качестве такого средства в *Maple* используются команды **op** и **nops**.

Эти команды по-разному действуют на объекты различного типа. Так, если анализируемый объект — выражение, то команда **nops** сообщает, сколько подобъектов (выражений) первого уровня находится в объекте, а команда **op** может использоваться, чтобы отобразить эти подобъекты в виде последовательности выражений. Приведем пример:

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

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

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

$$3$$

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

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

Команда **op** может также использоваться, чтобы извлечь индивидуальные элементы из объекта, а используемая рекурсивно, может забраться еще глубже — в подобъект.

```
> object:= x^3 * exp (1) - 34/Pi;
```

$$\mathrm{object} := x^3\;\mathbf{e} - \frac{34}{Pi}$$

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

$$x^3 \, e$$

$$x^3$$

```
> whattype (op (2, x^2+exp (1)-3));
```

function

Освоившись с командами **op** и **nops**, можно создавать более сложные команды для манипулирования элементами объекта. Например, можно создать команду, которая будет извлекать последний элемент выражения “object” (созданного ранее).

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

$$-\frac{34}{Pi}$$

Если исследуемый объект — индексная переменная (например с именем Iname), то команда **nops**(Iname) возвращает число индексов, **op**(i,Iname) возвращает i-ый индекс, а **op**(0,Iname) возвращает имя индексной переменной.

```
> nops(A[i,j]);nops(A[i][j]);
```

2

1

```
> op(1,A[i,j]);op(1,A[i][j]);
```

i

j

```
> op(0,A[i][j]);
```

$$A_i$$

Если исследуемый объект — функция-процедура (с именем Fname(x1,x2...)), то команда **nops** возвращает число аргументов этой функции, команда **op**(i,Fname(x1,x2,...)) возвращает i-тый аргумент, а команда **op**(0,Fname(x1, x2,...) — имя функции.

```
> nops(F(x,y,z));op(1..3,F(x,y,z));op(0,F(x,y,z));
```

3

x, y, z

F

5. Команды Maple

Изучив выражения *Maple* и такие типы объектов, как последовательности, списки и наборы, мы можем приступить к изучению команд *Maple*, использующих эти объекты в качестве параметров. Как упоминалось ранее, *Maple* имеет большое количество (более чем 2500) встроенных команд, сохраняемых в ядре *Maple*, основной библиотеке и специализированных пакетах.

Имена команд выбирались таким образом, чтобы лучше всего отразить функциональные возможности команд и, в то же время, быть как можно короче. Например, команда для интегрирования по частям называется **intparts**, а команда для замены переменных называется **changevar**. Некоторые имена команд длиной в один символ (например **D**), в то время как другие имеют длину более десяти символов (например **completesquare**). Большинство команд *Maple* написаны полностью в символах нижнего регистра. Однако следует помнить, что *Maple* "чувствителен к регистру". Это означает, что "diff" и "Diff" различны. Независимо от того используются ли они как имена команды или как имена переменных, *Maple* всегда рассматривает их как различные объекты.

Имена собственных команд *Maple* защищены. Это означает, что если вы попытаетесь использовать их для присвоения каких либо-выражений, то получите предупреждение об ошибке.

5.1. Последовательности параметров

Каждая команда *Maple* использует последовательность параметров в качестве аргумента.

```
> Имя_команды(параметр1, параметр2,...);
```

Эта последовательность может содержать несколько чисел, выражений, наборов или списков или может вообще не содержать никаких параметров. Независимо от того сколько параметров задано, параметры команды всегда заключены в круглые скобки ( ). Другие типы скобок не будут приводить к интерпретации записанного выражения как команды. Любой из рассмотренных объектов *Maple* (и некоторые из тех, которые еще не рассматривались) могут использоваться как параметры. Команды могут также использоваться как параметры; эти команды выполняются, и их результаты вставляются в последовательность параметров. Некоторые команды имеют ограничения на тип объектов, которые они применяют для ввода, и для большинства команд порядок параметров также важен. Все команды обязаны иметь минимальное число параметров, с которыми они могут вызываться (например команда **int** должна иметь, по крайней мере, два параметра — выражение и переменную интегрирования). В то же время большинство команд могут использовать больше параметров, чем их минимальное число. Эти экстра-параметры могут включать большое количество дополнительных возможностей, в том числе, опции управляющие функционированием команды.

Рассмотрим несколько примеров команд:

```
> Diff (3 *x^2+2* x-6, x);
```

$$\frac{d}{dx}(3\,x^2 + 2\,x - 6)$$

```
> diff (3 *x^2+2* x-6, x, x);
```

$$6$$

```
> trigexpr:=cos(x)^5 + sin(x)^4 + 2*cos(x)^2 -
  2*sin(x)^2 - cos(2*x);
> simplify(trigexpr);
```

$$trigexpr := \cos(x)^5 + \sin(x)^4 + 2\cos(x)^2 - 2\sin(x)^2 - \cos(2\,x)$$

$$\cos(x)^5 + \cos(x)^4$$

```
> Int ( Int ( x^2 * y^3, x ), y);value(");
```

$$\iint x^2\,y^3\,dx\,dy$$

$$\frac{1}{12}x^3\,y^4$$

5.2. Как вызвать команду?

Не всегда достаточно просто знать имя команды, которую вы хотите ввести — иногда вы должны загрузить команду из некоторой конкретной части библиотеки *Maple*. Если при вызове некоторой команды просто повторяется ввод, однако команда не выполняется, то это может означать, что вызываемая команда не существует или не загружена в память.

Приведем несколько примеров такого поведения.

```
> INT ( x^2, x );
```

$$\mathrm{INT}(x^2\,,x)$$

```
> ifactors( 120 );
```

$$\mathrm{ifactors}(-120)$$

```
> mean( 1, 2, 3, 4, 5, 6 );
```

$$\mathrm{mean}(1, 2, 3, 4, 5, 6)$$

Когда это случается, проверьте по буквам правильность записи команды (включая соответствие нижнего и верхнего регистров символов) и загрузили ли Вы команду в память *Maple.*

Автоматически загружаемые и загружаемые из библиотек команды

Когда программа *Maple* запускается, она не имеет ни одной команды, полностью загруженной в память. Однако большое количество стандартных команд имеют указатели их нахождения при загрузке. Когда вы вызываете одну из них, *Maple* загружает ее автоматически. Другие команды, постоянно находящиеся в основной библиотеке, автоматически не загружаются, а должны вначале явно загружаться командой **readlib** (чтение из библиотеки). Если при попытке вызвать команду из основной библиотеки команда не выполняется, следует перед командой поставить **readlib**.

Приведем некоторые примеры как автоматически загружаемых, так и загружаемых при помощи **readlib** команд.

```
> expand (( x-2) * (x+5));
```

$$x^2 + 3x - 10$$

```
> readlib(ifactors);
```

proc(n) ... end

```
> ifactors(120);
```

[1, [[2, 3], [3, 1], [5, 1]]]

Однажды загруженную в память команду нет необходимости перезагружать в течение текущего сеанса.

Команды в пакетах

Maple содержит несколько десятков специализированных наборов команд называемых пакетами (например **linalg**, **liesymm** и т.д.). Подпрограммы в этих пакетах не загружаются автоматически, и не могут инициироваться командой **readlib**. Один из способов вызова этих команд состоит в использовании команды **with**(имя пакета) для загрузки указателей ко всем командам пакета. Тогда при вызове любой команды пакета она автоматически загружается в память. Другой способ состоит в том, чтобы перед названием команды добавлять название пакета, а саму команду заключать в квадратные скобки. Следующие примеры иллюстрируют эти методы.

```
> with(combinat);
Warning, new definition for Chi
```

[Chi, bell, binomial, cartprod, character, choose, composition, conjpart, decodepart, encodepart, fibonacci, firstpart, graycode, inttovec, lastpart, multinomial, nextpart, numbcomb, numbcomp, numbpart, numbperm, partition, permute, powerset, prevpart, randcomb, randpart, randperm, stirling1, stirling2, subsets, vectoint]

```
> numbperm ([1,2,3,4]);
```

$$24$$

```
> with(stats):
```

```
> describe [mean] ([1,2,3,4,5,6]);
```

$$\frac{7}{2}$$

5.3. Некоторые часто используемые команды

Команд и функций, входящих в основную библиотеку около пятисот, часть из них — элементарные и специальные функции, другие предназначены для определения функций, процедур, операторов и других структур данных, большая часть команд предназначена для проведения вычислений с числами, многочленами, векторами и матрицами, функциями и другими объектами *Maple*, в том числе для дифференцирования и интегрирования, решения алгебраических и дифференциальных уравнений. Много команд предназначенных для графического представления выражений, для взаимодействия с системой, ввода и вывода информации и так далее. Трудно перечислить не только все команды, но даже все области применимости команд. Поэтому здесь мы рассмотрим наиболее общие, наиболее часто используемые команды, входящие в ядро *Maple* и основную библиотеку. Команды из специализированных пакетов будут рассмотрены в разделе “Обзор специализированных пакетов”.

Преобразование выражений

Прежде чем использовать команды *Maple* для выполнения алгебраических преобразований, вначале покажем, как выполняются стандартные преобразования без использования этих команд.

В качестве примера решим уравнение

```
> eq:=4*x+17=23;
```

$$eq := 4\,x + 17 = 23$$

Сначала вычтем из обеих частей уравнения 17

```
> eq-(17=17);
```

$$4\,x = 6$$

Теперь разделим уравнение на 4, чтобы получить ответ

```
> "/4;
```

$$x = \frac{3}{2}$$

Части выражения (команды lhs, rhs, numer, denom, remove, has, select, indet, subs, subsop)

Помимо рассмотренных выше команд **op**, **subsop** и **nops** существует еще несколько команд для оперирования элементами объектов **Maple**.

Рассмотрим уравнение

```
> eq:=a^2+b^2=c^2;
```

$$eq := a^2 + b^2 = c^2$$

Чтобы выделить левую или правую часть этого уравнения, существуют команды **lhs()** и **rhs()**.

```
> lhs(eq);
```

$$a^2 + b^2$$

```
> rhs(eq);
```

$$c^2$$

Эти же команды используются для выделения границ диапазона, задаваемого оператором диапазона i..j:

```
> lhs(2..5);i:=2..5;
```

$$2$$

$$i := 2\,..\,5$$

```
> rhs(2..5);
```

$$5$$

Существует также команды выделения числителя и знаменателя дробного выражения. Пусть имеется дробь

```
> fract:=(1+sin(x)^3-y/x)/(y^2-1+x);
```

$$fract := \frac{1 + \sin(x)^3 - y/x}{y^2 - 1 + x}$$

Можно из этой дроби выделить числитель командой **numer**:

```
> numer(fract);
```

$$x + \sin(x)^3\,x - y$$

или знаменатель командой **denom**:

```
> denom(fract);
```

$$x\,(y^2 - 1 + x)$$

Команда **remove** удаляет элементы из списка, набора, суммы, произведения или функции по заданному логическому отношению (правилу).

Например, зададим правило в виде :

```
> large:=z->evalb(is(z>3)=true);
```

$$large := z \to \mathrm{evalb}(\mathrm{is}(3 < z) = true)$$

Теперь по этому правилу удалим элементы из выражения

```
> remove(large,5+8*sin(x)-exp(9));
```

$$8\sin(x) - \mathbf{e}^9$$

Команда **has** проверяет наличие данного операнда в выражении. Пусть

```
> f := 2*exp(a*x)*sin(x)*ln(y);
```

$$f := 2\,\mathbf{e}^{(a\,x)}\sin(x)\ln(y)$$

```
> has(f,exp(a*x));
```

$$true$$

Команда **select** извлекает из выражения операнды заданного типа

```
> select(has, f, x);
```

$$\mathbf{e}^{(a\,x)}\sin(x)$$

```
> remove(has, f, x);
```

$$2\ln(y)$$

Команда **indets** перечисляет все независимые переменные, находящиеся в математическом выражении, или переменные заданного типа

```
> indets(f);indets(f,`function`);
```

$$\{x, y, a, \sin(x), \mathbf{e}^{(a\,x)}, \ln(y)\}$$

$$\{\sin(x), \mathbf{e}^{(a\,x)}, \ln(y)\}$$

Команда подстановки **subs**(a=b, выражение) позволяет заменить в математическом выражении один операнд (a) другим (b), в частности числом. Приведем пример, поясняющий действие этой команды.

```
> restart;
> y:=ln(sin(x*exp(cos(x))));
```

$$y := \ln(\sin(x\, e^{(\cos(x))}))$$

```
> yprime:=diff(y,x);
```

$$yprime := \frac{\cos(x\, e^{\cos(x)})\,(e^{\cos(x)} - x\sin(x)\, e^{\cos(x)})}{\sin(x\, e^{\cos(x)})}$$

```
> subs(x=2,yprime);
```

$$\frac{\cos(2\, e^{\cos(2)})\,(e^{\cos(2)} - 2\sin(2)\, e^{\cos(2)})}{\sin(2\, e^{\cos(2)})}$$

```
> evalf(");
```

$$-.1388047428$$

Команда **subsop**(n=b, выражение) позволяет заменить n-тый операнд выражения (n-ый аргумент функции, n-ый элемент списка) на операнд b.

```
> expr:=cos(x)+exp(b*y);
```

$$expr := \cos(x) + e^{(b \ln(\sin(x\, e^{\cos(x)})))}$$

```
> subsop(1=t,expr);
```

$$t + e^{(b \ln(\sin(x\, e^{\cos(x)})))}$$

Название функции — нулевой операнд и его также можно заменить командой **subsop()**:

```
> f1:=subsop(0=sin,Dirac(x));subs(x=Pi/4,f1);
```

$$f1 := \sin(x)$$

$$\sin\left(\frac{1}{4}\pi\right)$$

Команда simplify

Рассмотрим арифметический корень

```
> restart;expr:=sqrt((x*y)^2);
```

$$expr := \sqrt{x^2\, y^2}$$

Вообще говоря, он не упрощается, поскольку ответ будет зависеть от области изменения x и y.

```
> simplify(expr);
```

$$\sqrt{x^2\, y^2}$$

Мы можем ввести опцию, указывающую, что x и y действительны

```
> simplify(expr,assume=real);
```

$$\mathrm{signum}(x)\; x\; \mathrm{signum}(y)\; y$$

либо положительны

```
> simplify(expr,assume=positive);
```

$$x\, y$$

Рассмотрим еще одно выражение

```
> expr:=x*y*z+x*y+x*z+y*z;
```

$$expr := x\,y\,z + x\,y + x\,z + y\,z$$

можно упростить его, задав в виде условия некоторое соотношение между переменными

```
> simplify(expr,{x*z=1});
```

$$x\,y + y + 1 + y\,z$$

Мы можем явно в команде **simplify** указать вид замены в выражении. Рассмотрим пример:

```
> expr1:=x^3+y^3;
```

$$expr1 := x^3 + y^3$$

```
> siderel:=x^2+y^2=1;
```

$$siderel := x^2 + y^2 = 1$$

```
> simplify(expr1,{siderel});
```

$$x^3 - y\,x^2 + y$$

В третьем параметре аргумента команды **simplify** мы можем указать также порядок замены. Например, заменить x на y

```
> simplify(expr1,{siderel},[x,y]);
```

$$y^3 + x - x\,y^2$$

или наоборот y на x.

```
> simplify(expr1,{siderel},[y,x]);
```

$$x^3 - y\,x^2 + y$$

Как видим, результаты различны.

Команды expand и factor

Командой **expand** мы инициируем выполнение умножения

```
> expand((x+1)*(y+z),x+1);
```

$$(x + 1)\,y + (x + 1)\,z$$

```
> expand((x+1)*(y+z));
```

$$x\,y + x\,z + y + z$$

```
> exp(a+ln(b));
```

$$\mathbf{e}^{(a + \ln(b))}$$

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

$$\mathbf{e}^a\,b$$

```
> expand(exp(a+ln(b)));
```

$$\mathbf{e}^a\,b$$

Наоборот, команда **factor** позволяет разложить многочлен или рациональную дробь на множители

```
> big_poly:=x^5-x^4-7*x^3+x^2+6*x;
```

$$big_poly := x^5 - x^4 - 7\,x^3 + x^2 + 6\,x$$

```
> factor(big_poly);
```

$$x\,(x - 1)\,(x - 3)\,(x + 2)\,(x + 1)$$

```
> rat_expr:=(x^3-y^3)/(x^4-y^4);
```

$$rat_expr := \frac{x^3 - y^3}{x^4 - y^4}$$

```
> factor(rat_expr);
```

$$\frac{y^2 + x\,y + x^2}{(x + y)\,(x^2 + y^2)}$$

```
> roots(big_poly);
```

$$[[1, 1], [-2, 1], [3, 1], [0, 1], [-1, 1]]$$

Команда normal

Эта команда позволяет сократить рациональную дробь

```
> normal(rat_expr,'expanded');
```

$$\frac{y^2 + x\,y + x^2}{y^3 + x\,y^2 + x^2\,y + x^3}$$

Команда combine

Эта команда пытается объединить показатели степенных функций и понизить степень тригонометрических выражений.

```
> combine((x^a)^2,power);
```

$$x^{(2\,a)}$$

```
> combine(4*sin(x)^3,trig);
```

$$-\sin(3\,x) + 3\sin(x)$$

Команда assume

При помощи этой команды мы накладываем некоторые ограничения на переменную

```
> assume(m>-10);
```

Если нам необходимо ввести дополнительные ограничения, то используется команда

```
> additionally(m<=0);
```

Теперь мы можем вызвать описание переменной

```
> about(m);
Originally m, renamed m~:
  is assumed to be: RealRange(Open(-10),0)
```

```
> frac(n);
```

$$\mathrm{frac}(n)$$

```
> assume(n,integer);
```

```
> frac(n);
```

$$0$$

Попробуем вычислить интеграл.

```
> int(exp(c*x),x=0..infinity);
Definite integration: Can't determine if the integral is
convergent.
Need to know the sign of --> -c
Will now try indefinite integration and then take limits.
```

$$\lim_{x \to \infty} \frac{\mathrm{e}^{(c\,x)}}{c} - \frac{1}{c}$$

Maple выдает ошибку, которая означает, что невозможно определить, будет ли расходиться интеграл, если не указан знак параметра с. Наложим ограничение на с:

```
> assume(c<0);
```

```
> int(exp(c*x),x=0..infinity);
```

$$-\frac{1}{c\sim}$$

Как видим, теперь интегрирование выполняется. Вообще все переменные в *Maple* по умолчанию считаются комплексными. И почти все алгебраические функции умеют оперировать с комплексными величинами, например

```
> ln(exp(3*Pi*I));
```

$$I\,\pi$$

Рассмотрим уравнение

```
> eq:=xi^2=a;
```

$$eq := \xi^2 = a$$

наложим ограничение на а

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

и решим уравнение

```
> solve(eq,{xi});
```

$$\{\xi = I\sqrt{-a\text{~}}\}, \{\xi = -I\sqrt{-a\text{~}}\}$$

Попробуем отменить ограничение подстановкой

```
> eq:=subs(a='a',eq);
```

$$eq := \xi^2 = a$$

```
> solve(eq,{xi});
```

$$\{\xi = I\sqrt{-a\text{~}}\}, \{\xi = -I\sqrt{-a\text{~}}\}$$

Тильда ~ после **a** указывает, что такая отмена не сработала.

```
> a:=evaln(a);
```

$$a := a$$

```
> solve(eq,{xi});
```

$$\{\xi = \sqrt{a}\}, \{\xi = -\sqrt{a}\}$$

Теперь все в порядке. Для отмены ограничений достаточно либо ввести новую команду assume, либо использовать команду отмены присвоения **evaln**, либо кавычки.

Команды map, add, mul

map — очень полезная команда, она позволяет применить функцию к каждому из элементов списка, набора или к операндам верхнего уровня выражения. Если команда применена к списку, то ее результатом оказывается список функций, примененных к каждому из элементов списка в том же порядке, в котором они расположены в списке. Применим команду **map**, в аргументе которой первым параметром поставим некоторую функцию f, а вторым список [a,b,c]:

```
> map(f,[a,b,c]);
```

$$[f(a), f(b), f(c)]$$

Мы видим, что команда **map** направляет действие функции на каждый из элементов списка. Если, например, функция f возводит в квадрат, то

```
> map(x->x^2,[a,b,c]);
```

$$[a^2, b^2, c^2]$$

В команду **map** возможно также введение третьего и последующих параметров. В этом случае функция также превратиться в список функций от соответствующего числа параметров, причем параметры аргумента команды, стоящие за вторым, добавляются вслед за параметром списка в аргументы каждой из списка функций.

```
> map(f,[a,b,c],p,q);
```

$$[f(a, p, q), f(b, p, q), f(c, p, q)]$$

Существует также команда **map2**, в которой список элементов является третьим аргументом, а вторым — дополнительный элемент. Эта команда также применяет функцию к каждому из элементов списка, однако дополнительный элемент располагается перед параметром списка в аргументы каждой из списка функций.

```
> map2(f,p,[a,b,c]);
```

$$[f(p, a), f(p, b), f(p, c)]$$

Эта команда также позволяет вводить дополнительные параметры вслед за параметром-списком, которые помещаются в аргументы результирующего списка функций вслед за элементами списка:

```
> map2(f,p,[a,b,c],q,r);
```

$$[f(p, a, q, r), f(p, b, q, r), f(p, c, q, r)]$$

В качестве примера функции от двух параметров приведем операцию дифференцирования

```
> map(diff,[(x+1)*(x+2),x*(x-1)],x);
```

$$[2x + 3, 2x - 1]$$

Команда **map2** посылает элементы списка на второй параметр аргумента команды **diff**, что позволяет получить список производных по каждой из трех переменных списка.

```
> map2(diff,x^y/z,[x,y,z]);
```

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

Сама функция, являющаяся первым параметром аргумента команды **map**, может быть списком или набором функций. В этом случае все функции данного набора применяются к последующим параметрам аргумента команды.

```
> map({[a,b],[c,d],[e,f]},p,q);
```

$$\{[a(p, q), b(p, q)], [e(p, q), f(p, q)], [c(p, q), d(p, q)]\}$$

```
> map2(map,{[a,b],[c,d],[e,f]},p,q);
```

$$\{[a(p, q), b(p, q)], [e(p, q), f(p, q)], [c(p, q), d(p, q)]\}$$

На следующем примере показано, как команда **map** направляет функцию на операнды выражения верхнего уровня

```
> map(f,(x+y)/sin(z));
```

$$f(x + y)\, f\left(\frac{1}{\sin(z)}\right)$$

Те же преобразования, которые выполняет команда **map**, возможны также при помощи команды **seq**.

```
> seq(f(i),i={a,b,c});
```

$$f(a), f(b), f(c)$$

```
> seq(f(p,i,q,r),i=[a,b,c]);
```

$$f(p, a, q, r), f(p, b, q, r), f(p, c, q, r)$$

```
> seq(diff(j,x),j=[(x+1)*(x+2),x*(x-1)]);
```

$$2\,x + 3, 2\,x - 1$$

```
> seq(diff(x^y/z,k),k=[x,y,z]);
```

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

Еще одна очень полезная команда **add**(), которая позволяет применять функцию к элементам списка с последующим суммированием.

```
> add(i^2,i=[5,y,sin(x),-5]);
```

$$50 + y^2 + \sin(x)^2$$

```
> L := [seq(i, i=1..5)];
```

$$L := [1, 2, 3, 4, 5]$$

```
> add( (x+i)^2, i=L );
```

$$(x + 1)^2 + (x + 2)^2 + (x + 3)^2 + (x + 4)^2 + (x + 5)^2$$

Аналогичная команде **add** команда **mul** выполняет умножение.

```
> mul( x-i, i=L );
```

$$(x-1)(x-2)(x-3)(x-4)(x-5)$$

Изменение типа выражения (команда convert)

Команда *Maple* **convert**(выражение, тип) позволяет конвертировать тип выражения в другой тип, или, иначе говоря, изменять тип выражения. Дело в том, что многие команды *Maple* рассчитаны на использование с выражениями только определенных типов. Меняя тип выражения, мы получаем возможность применять к данному выражению ранее не выполнявшиеся команды.

Например, если разложить функцию sin(x) в ряд Тейлора, то мы получим выражение типа **series.**

```
> f:=sin(x);
```

$$f := \sin(x)$$

```
> t:=taylor(f,x=0);whattype(t);
```

$$t := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^6)$$

series

Однако, чтобы получать приближенные численные значения этого ряда, необходимо конвертировать его в полином:

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

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

$$+$$

Чтобы вывести полученный полином в название графика, необходимо конвертировать его в строку

```
> p_txt:=convert(p,string);
```

*p_txt := x − 1/6 * x^3 + 1/120 * x^5*

Теперь мы можем построить график полинома (рис. 6).

```
> plot({f,p},x=-4..4,title=p_txt);
```

x-1/6*x^3+1/120*x^5

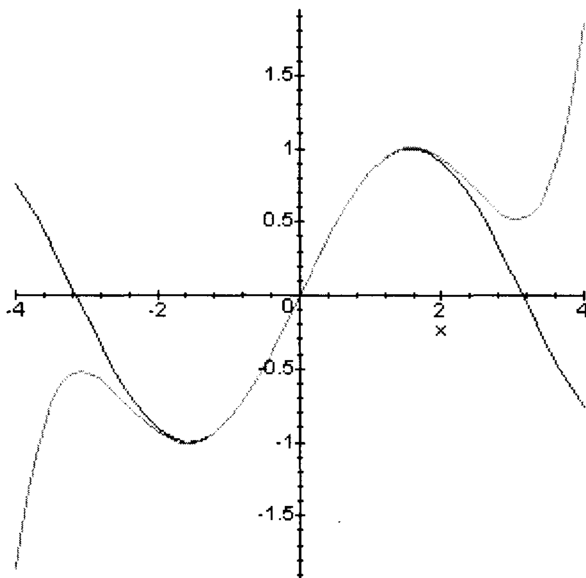

Рис. 6

Для исключения одинаковых элементов из списка можно конвертировать его в набор (в котором одинаковые элементы автоматически удаляются), а затем обратно в список:

```
> L:=[1,2,5,2,1];
```

$$L := [1, 2, 5, 2, 1]$$

```
> S:=convert(L,set);
```

$$S := \{1, 2, 5\}$$

```
> convert(S,list);
```

$$[1, 2, 5]$$

Приведем еще несколько примеров

```
> convert(cos(x),exp);
```

$$\frac{1}{2}\, e^{(Ix)} + \frac{1}{2}\, \frac{1}{e^{(Ix)}}$$

```
> convert(1/2*exp(x)+1/2*exp(-x),trig);
```

$$\cosh(x)$$

6. Примеры вычислений

В этом разделе кратко описаны вычислительные "способности" программы *Maple* при решении ряда математических задач в интерактивном режиме. Здесь проиллюстрирована только очень малая доля того, что умеет *Maple*, тем не менее, читатель из полученных примеров может получить некоторые навыки проведения вычислений в среде *Maple*.

6.1. Преобразование алгебраических выражений

Многочлены и рациональные дроби

```
> restart;pol:=x^4-x^3-11*x^2+31*x-20;
```

$$pol := x^4 - x^3 - 11x^2 + 31x - 20$$

```
> factor(");
```

$$(x - 1)\ (x + 4)\ (x^2 - 4x + 5)$$

```
> factor(x^25+1);
```

$$(x + 1)\ (1 - x + x^2 - x^3 + x^4)\ (1 - x^5 + x^{10} - x^{15} + x^{20})$$

```
> expr:=x^2*(y-z)+y^2*(z-x)+z^2*(x-y);
```

$$expr := x^2\ (y - z) + y^2\ (z - x) + z^2\ (x - y)$$

```
> factor(expr);
```

$$(y - z)\ (-z + x)\ (x - y)$$

Следующий пример иллюстрирует упрощение алгебраического выражения **expr** при заданных соотношениях **constr1**, **constr2**, связывающих переменные

```
> expr :=(-c^3-b^3+a^3)^2*(y^2+x^2)^2;
```

$$expr := (-c^3 - b^3 + a^3)^2\ (y^2 + x^2)^2$$

```
> constr1 := y^2+x^2 = m; constr2 := -c^3-b^3+a^3 = 
  3*n;
```

$$constr1 := y^2 + x^2 = m$$

$$constr2 := -c^3 - b^3 + a^3 = 3n$$

```
> simplify(expr,{constr1,constr2});
```

$$9\ m^2\ n^2$$

Сложные радикалы

Maple эффективно упрощает сложные радикалы, например

$$> \sqrt{14 + 3\sqrt{3 + 2\sqrt{5 - 12\sqrt{3 - 2\sqrt{2}}}}}$$

$$3 + \sqrt{2}$$

Отметим, что упрощение данного радикала было автоматическим, без применения команды **simplify**.

Для упрощения выражений, содержащих не только квадратные корни, но и радикалы других степеней, применяется команда **radnormal**. Приведем примеры

```
> sqrt(3+sqrt(3)+(10+6*sqrt(3))^(1/3))= radnormal
  (sqrt (3+sqrt(3)+(10+6* sqrt(3))^(1/3)));
```

$$\sqrt{3 + \sqrt{3} + (10 + 6\sqrt{3})^{1/3}} = 1 + \sqrt{3}$$

```
> (4+3*3^(2/3)+3*3^(1/3))^(1/3)=radnormal((4+3*3^
  (2/3)+3*3^(1/3))^(1/3));
```

$$(4 + 3\ 3^{2/3} + 3\ 3^{1/3})^{1/3} = 1 + 3^{1/3}$$

Тригонометрические выражения

```
> x:=a*cos(alpha)*sin(beta);
  y:=a*sin(alpha)*sin(beta);
  z:=a*cos(beta);
```

$$x := a\cos(\alpha)\sin(\beta);$$

$$y := a\sin(\alpha)\sin(\beta);$$

$$z := a\cos(\beta);$$

```
> simplify(x^2+y^2+z^2);
```

$$a^2$$

В некоторых случаях для эффективного упрощения тригонометрических выражений приходится применять некоторые ухищрения. Рассмотрим, например, выражение

```
> B:=(3-4*cos(2*alpha)+cos(4*alpha))/(3+4*cos(2*alpha)
  + cos(4*alpha));
```

$$\frac{3-4\cos(2\,\alpha)+\cos(4\,\alpha)}{3-4\cos(2\,\alpha)+\cos(4\,\alpha)}$$

Простая команда **simplify** не приводит к существенному упрощению

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

$$\frac{1-2\cos(2\,\alpha)+\cos(2\,\alpha)^2}{1+2\cos(2\,\alpha)+\cos(2\,\alpha)^2}$$

Преобразуем вначале косинусы в тангенсы

```
> convert(",tan);
```

$$\frac{1-2\,\dfrac{1-\tan(\alpha)^2}{1+\tan(\alpha)^2}+\dfrac{(1-\tan(\alpha)^2)^2}{(1+\tan(\alpha)^2)^2}}{1+2\,\dfrac{1-\tan(\alpha)^2}{1+\tan(\alpha)^2}+\dfrac{(1-\tan(\alpha)^2)^2}{(1+\tan(\alpha)^2)^2}}$$

А после этого упростим

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

$$\tan(\alpha)^4$$

6.2. Решение уравнений и неравенств

В этом разделе будет рассмотрена методика решения уравнений, неравенств и систем и проверки правильности решения.

Решение систем уравнений

Рассмотрим систему уравнений

```
> eqns:={x+2*y=3,y+1/x=1};
```

$$eqns:=\{x+2\,y=3,\ y+\frac{1}{x}=1\}$$

Решение находится командой **solve**. Присвоим решению имя soln.

```
> soln:=solve(eqns,{x,y});
```

$$soln := \{x = -1,\ y = 2\},\ \{x = 2,\ y = \frac{1}{2}\}$$

Мы видим, что решение представляет из себя два набора уравнений. Можно выделить каждый из них

```
> soln[1];
```

$$\{x = -1,\ y = 2\}$$

```
> soln[2];
```

$$\{x = 2,\ y = \frac{1}{2}\}$$

Проверка решения

Проверка осуществляется подстановкой решений в исходную систему

```
> subs(soln[1],eqns);
```

$$\{1 = 1,\ 3 = 3\}$$

```
> subs(soln[2],eqns);
```

$$\{1 = 1,\ 3 = 3\}$$

В то же время переменным x и y не присвоены никакие значения. Присвоим переменным {x1, y1}, {x2, y2} решения 1 и 2 соответственно. Это делается при помощи команды **subs**

```
> x1:=subs(soln[1],x);
```

$$x1 := -1$$

```
> x2:=subs(soln[2],x);
```

$$x2 := 2$$

```
> y1:=subs(soln[1],y);
```

$$y1 := 2$$

```
> y2:=subs(soln[2],y);
```

$$y2 := \frac{1}{2}$$

Эту команду можно использовать также для других подстановок решения

```
> subs(soln[1],[x,y]);
```

$$[-1, 2]$$

```
> [soln];subs(soln,eqns);
```

$$[\{x = -1, y = 2\}, \{x = 2, y = \frac{1}{2}\}]$$

$$\{1 = 1, 3 = 3\}$$

Системы линейных уравнений

Рассмотрим линейную систему из пяти уравнений

```
> eqn1:=x+2*y+3*z+4*t+5*u=41:
> eqn2:=5*x+5*y+4*z+3*t+2*u=20:
> eqn3:=3*y+4*z-8*t+2*u=125:
> eqn4:=x+y+z+t+u=9:
> eqn5:=8*x+4*z+3*t+2*u=11:
```

Можно получить решение системы трех из этих уравнений для трех переменных. В этом случае решения будут функциями от остальных переменных

```
> s2:=solve({eqn1,eqn2,eqn3},{x,y,z});
```

$$s2 := \{y = 12\ t + \frac{70}{13}u + \frac{635}{13},\ x = -7\ t - \frac{28}{13}u - \frac{527}{13},\ z = -7\ \mathrm{t} - \frac{59}{13}\mathrm{u} - \frac{70}{13}$$

Чтобы найти решение для конкретных значений u и t, можно сделать подстановку, например

```
> subs({u=1,t=1},s2);
```

$$\{y = \frac{861}{13},\ x = \frac{-646}{13},\ z = \frac{-220}{13}\}$$

Можно также решить систему из пяти уравнений для пяти неизвестных

```
> s1:=solve({eqn1,eqn2,eqn3,eqn4,eqn5}, {x,y,z,t,u});
```

$$s1 : \{z = -1,\ y = 3,\ u = 16,\ t = -11,\ x = 2\}$$

В этом случае решение единственно.

Чтобы получить представление о всех решениях s2, создадим следующей подстановкой список решений.

```
> subs(s2,[x,y,z]);
```

$$-7\,t-\frac{28}{13}\,u-\frac{527}{13},\quad 12\,t+\frac{70}{13}\,u+\frac{635}{13},\ -7\,\mathrm{t}-\frac{59}{13}\,\mathrm{u}-\frac{70}{13}$$

Теперь можно построить график поверхности, являющейся решением s2 (рис. 7).

```
> plot3d(",u=0..2,t=0..2);
```

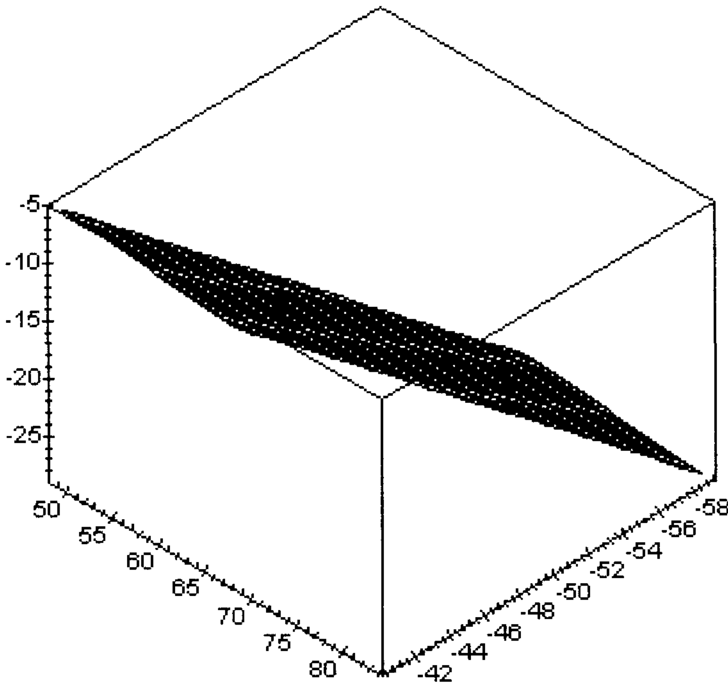

Рис. 7

Корни многочленов

При помощи функции **randpoly** создадим полином 40-ой степени со случайными коэффициентами (коэффициенты — целые случайные числа в интервале −100..100):

```
> restart;poly1:=randpoly([x], degree = 40);
```

$$poly1 : 79\, x^{29} + 56\, x^{23} + 49\, x^{18} + 63\, x^{16} + 57\, x^{10} + 59\, x^{3}$$

Найдем корни полинома командой **solve**:

```
> S:=solve(poly1,x);
```

$$S : 0, 0, 0, \text{Root Of}(79\ _Z^{26} + 56\ _Z^{20} + 49\ _Z^{15} + 63\ _Z^{13} + 57\ _Z^{7} - 59$$

В общем случае явных решений в виде радикалов для корней полиномов степени выше четвертой не существует. В этом случае дается неявное решение в виде *RootOf* (poly). В некоторых случаях *Maple* может найти явное решение для корней, однако выводит на дисплей неявное решение в виде тех же *RootOf*. Чтобы инициировать вывод явных решений, можно:

- присвоить переменной операционной среды *_EnvExplicit* значение true;
- конвертировать выражения, содержащие *RootOf*, в радикалы командой **convert**(expr, radical).

Чтобы получить приближенное значение всех корней в *Maple* используется команда **allvalues**, раскрывающая структуру *RootOf*:

```
> Sapr:=[S[1],S[2],S[3],allvalues(S[4])];
```

Sapr := [0, 0, 0, −1.036781034, −.9200669289 −.5244675161 *I*,

−.9200669289 + .5244675161 *I*, −.8935692255 −.2610071369 *I*,

−.8935692255 − .2610071369 *I*, −.7664243339 −.5729068937 *I*,

−.7664243339 −.5729068937 *I*, −.5611063377 −.8431685381 *I*,

−.5611063377 −.8431685381 *I*, −.3020685037 −.8896325835 *I*,

−.3020685037 −.8896325835 *I*, −.1139153281 −.9872232020 *I*,

−.1139153281 −.9872232020 *I*, .08670984550 −.1.026893462 *I*,

.08670984550 +.1.026893462 *I*, .3415049682 −.9389962939 *I*,

.3415049682 −.9389962939 *I*,

.5461489429 −.7275164660 *I*, .5461489429 +.7275164660 *I*,

7656237873 −.6994306831 *I*,

7656237873 +.6994306831 *I*, 8880289644 −.4577688178 *I*,

8880289644 +.4577688178 *I*,

.8926990381, 1.001175147 −.2725044079 *I*, 1.001175147 −.2725044079 *I*]

Теперь построим на комплексной плоскости все корни полинома (рис. 8).

```
> with(plots):complexplot(Sapr, x=-1.1..1.1, style=p-
oint);
```

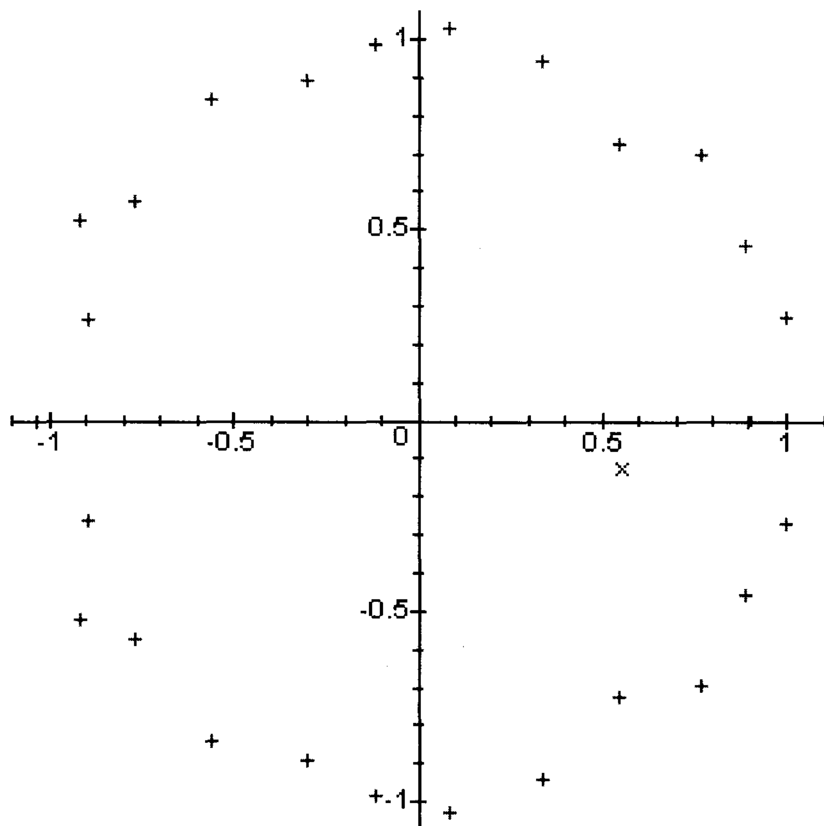

Рис. 8.

Интересный результат — почти все корни случайного полинома располагаются на комплексной плоскости вблизи окружности единичного радиуса с центром в начале координат.

Системы нелинейных уравнений

Рассмотрим систему из трех уравнений от переменных x, y, z.

```
> restart;
> f1 := a^4*z^2+2*a^3*y*z-2*a^2*x*z+a^2*y^2-2*a*x*y;
```

$$f1 := a^4 z^2 + 2\, a^3\, y\, z - 2\, a^2\, x\, z + a^2\, y^2 - 2\, a\, x\, y;$$

```
> f2 := a^4*z^2+3*a^3*y*z-a^2*x*z+3*a*x*y-2*x^2;
```

$$f2 := a^4 z^2 + 3\, a^3\, y\, z - a^2\, x\, z + -\, 3\, a\, x\, y - 2\, x^2;$$

```
> f3 := a^4*z^2+a^3*y*z-2*a^2*x*z-2*a*x*y;
```

$$f3 := a^4 z^2 + a^3\, y\, z - 2\, a^2\, x\, z - 2\, a\, x\, y;$$

Решение этой системы получим при помощи команды **solve**

```
> sols:=solve({f1,f2,f3},{x,y,z});
```

$$sols := \{x = \frac{1}{2}\, a^2\, z,\ z = z,\ y = 0\},\ \{z = z,\ x = -a^2\, z,\ y = -a\, z\}$$

Отметим, что в этом примере потеряно тривиальное решение {x=0, y=0, z=0}. Проверим остальные два

```
> map(subs,[sols[1],sols[2]],[f1,f2,f3]));
```

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

Еще один пример из четырех нелинейных уравнений

```
> eqs:={u*v*y^2=8,v*y^2*w=24,y^2*w*u= 12,u+v+w=y-
+4};
```

$$eqs := \{u\, v\, y^2 = 8, v\, y^2\, w = 24,\ y^2\, w\, u = 12,\ u + v + w = y + 4\};$$

Для получения явного решения в радикалах введем команду

```
> _EnvExplicit := true:
```

```
> soln:=solve(eqs,{y,u,v,w});
```

$$soln := \{y = 2,\ v = 2,\ u = 1,\ w = 3\},\ \ \{w = -1,\ v = \frac{-2}{3},\ u = \frac{-1}{3},\ y = -6\}$$

$$\{y = -2 + 2\ I\sqrt{2},\ v = \frac{2}{3} + \frac{2}{3}\ I\sqrt{2},\ w = 1 + I\sqrt{2},\ u = \frac{1}{3} + \frac{1}{3}\ I\sqrt{2}\}$$

$$\{y = -2 + 2\ I\sqrt{2},\ v = \frac{2}{3} + \frac{2}{3}\ I\sqrt{2},\ w = 1 + I\sqrt{2},\ u = \frac{1}{3} + \frac{1}{3}\ I\sqrt{2}\}$$

Решение рекуррентных и функциональных уравнений

Начнем с примера

```
> rsolve({y(n)*y(n-1) + y(n) - y(n-1) = 0,
  y(0)=a}, y);
```

$$\frac{a}{1 + na}$$

Отметим, что команда **rsolve** фактически позволяет решить функциональное уравнение для целой функции y(n). Приведем еще один пример

```
> rec:=g(n)-g(2*n) = 1+1/n;
```

$$rec := g(n) - g(2\ n) = 1 + \frac{1}{n};$$

```
> rsolve(rec,g);
```

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

Команда **solve** также позволяет решать функциональные уравнения, например

```
> F:=solve(f(x)^2-3*f(x)+2*x, f);
```

$$F := \textbf{proc}(x)\ \mathrm{RootOf}(_Z\hat{\ }2 - 3*_Z + 2*x)\ \textbf{end}$$

Можно преобразовать полученное неявное решение в радикалы

```
> convert(F(x),radical);
```

$$\frac{3}{2} + \frac{1}{2}\sqrt{9 - 8\ x}$$

Решение трансцендентных уравнений и систем

Уравнение, содержащее показательные функции

```
> eqn := x^2*2^x+8=2*x^2+2^(x+2);
```

$$eqn := x^2\,2^x + 8 = 2\,x^2 + 2^{(x+2)}$$

```
> solve(eqn,x);
```

$$2, -2, 1$$

Система уравнений, содержащая показательные и логарифмические функции

```
> restart;eqns:={7*3^x-3*2^(z+y-x+2)=15,
           2*3^(x+1)+3*2^(z+y-x)=66,
           ln(x+y+z)-3*ln(x)-ln(y*z)=-ln(4)};
```

$$eqns := \{\ln(x + y + z) - 3\ln(x) - \ln(y\,z) = -\ln(4),$$
$$2\,3^{(x+1)} + 3\,2^{(z+y-x)} = 66,\ 7\,3^x - 3\,2^{(z+y-x+2)} = 15\}$$

Мы не выводим на дисплей решение до команды **simplify** ввиду громоздкости.

```
> _EnvExplicit := true:
> soln:=solve(eqns,{x,y,z}):
> map(simplify,[soln[1],soln[2]]);
```

$$[\{x = 2, y = 3, z = 1\}, \{x = 2, y = 3, z = 1\}]$$

Решение тригонометрических уравнений

Начнем с примера

```
> solve( cos(x)+y = 9, x );
```

$$\pi - \arccos(y - 9)$$

Таким образом, мы находим только главное решение тригонометрического уравнения; чтобы найти все решения, необходимо ввести команду

```
> _EnvAllSolutions := true:
```

которая устанавливает для переменной операционной среды **_EnvAllSolutions** значение **true**. Теперь решение того же уравнения

```
> solve( cos(x)+y = 9, x );
```

$$\pi - \arccos(y-9) - 2\ _B1\sim\ \pi + 2\ _B1\sim \arccos(y-9) + 2\pi\ _Z1\sim$$

где _B~ и _Z~ — натуральные числа.

Пример посложнее

```
> restart;eqn:=sqrt(cos(x)^2+1/2)+sqrt(sin(x)^2+1/
  2)=2;
```

$$eqn := \frac{1}{2}\sqrt{4\cos(x)^2+2} + \frac{1}{2}\sqrt{4\sin(x)^2+2} = 2$$

```
> sols:=solve(eqn,x);
```

$$sols := \frac{1}{4}\pi + 2\pi\ _Z\sim,\ -\frac{3}{4}\pi + 2\pi\ _Z\sim$$

Решение неравенств

Приведем два примера

```
> solve( x^2+x>5, x );
```

$$\text{RealRange}\left(-\infty, \text{Open}\left(-\frac{1}{2} - \frac{1}{2}\sqrt{21}\right)\right),$$

$$\text{RealRange}\left(\text{Open}\left(-\frac{1}{2} - \frac{1}{2}\sqrt{21}\right), \infty\right)$$

```
> solve((x-1)*(x-2)*(x-3) < 0, x);
```

$$\text{RealRange}(-\infty, \text{Open}(1)),\ \text{RealRange}(\text{Open}(2), \text{Open}(3))$$

Maple умеет решать также трансцендентные неравенства, например

```
> uneqn := exp(x)*x^2 >= 1/2:
> solve( uneqn, {x} );evalf(");
```

$$\left\{2\ \text{LambertW}\left(-1, -\frac{1}{4}\sqrt{2}\right) \le x,\ x \le 2\ \text{LambertW}\left(-\frac{1}{4}\sqrt{2}\right)\right\},$$

$$\left\{2\ \text{LambertW}\left(\frac{1}{4} - \sqrt{2}\right) \le x\right\}$$

$$\{-2.617866616 \le x,\ x \le -1.487962064\},\ \{.5398352768 \le x\}$$

Следующий пример системы неравенств

```
> solve({x+y>=2,
         x-2*y<=1,
         x-y>=0,
         x-2*y>=1},
         {x,y});
```

$$\{x = 1 + 2\,y, \frac{1}{3} \le y\}$$

6.3. Нахождение экстремумов функций, симплекс-метод

```
> readlib(extrema): _Env Explicit := true:
```

В следующем примере находятся экстремумы функции f при дополнительных ограничениях g1 и g2:

```
> f := (x^2+y^2)^(1/2)-z; g1 := x^2+y^2-16=0;
  g2 := x+y+z = 10;
```

$$f := \sqrt{x^2 + y^2} - z$$

$$g1 := x^2 + y^2 - 16 = 0$$

$$g2 := x + y + z = 10$$

В команде **extrema** четвертым аргументом является имя, которому мы хотим присвоить значение переменных в точках экстремума.

```
> extrema(f,{g1,g2},{x,y,z},'s');
```

$$\{-6 + 4\sqrt{2}, -6 - 4\sqrt{2}\}$$

```
> s;
```

$$\{\{z = -4\sqrt{2} + 10, y = 2\sqrt{2}, x = 2\sqrt{2}\},$$
$$\{z = 4\sqrt{2} + 10, y = -2\sqrt{2}, x = -2\sqrt{2}\}\}$$

Для применения симплекс-метода необходимо загрузить пакет **simplex**

```
> restart;with(simplex):
Warning, new definition for maximize
Warning, new definition for minimize
```

В следующем примере максимизируется функция obj при дополнительных ограничениях **cnsts**:

```
> cnsts := {3*x+4*y-3*z <= 23, 5*x-4*y-3*z <= 10,
  7*x+4*y+11*z <= 30}:
  obj := -x + y + 2*z:
  maximize(obj,cnsts union {x>=0,y>=0,z>=0});
```

$$\{x = 0, z = 1/2, y = 49/8\}$$

6.4. Дифференцирование

Команда **diff(**параметр1, параметр2**)** обеспечивает дифференцирование выражения (первый параметр) по переменной (второй параметр). Аргумент команды должен содержать, по крайней мере, одну переменную дифференцирования, последующие параметры интерпретируются как переменные для дифференцирования более высокого порядка. Приведем пример

```
> restart;diff ( x^3 * y^2, x, y);
```

$$6\, x^2\, y$$

Для выполнения многократного дифференцирования по одной переменной для сокращения записи можно использовать в команде **diff** оператор **$**.

```
> diff ( x^6/6!, x$6 );
```

$$1$$

```
> diff ((s^3+2*s-5)/(t^2-3*t), s$2, t);
```

$$-6\, \frac{s\,(2\,t - 3)}{(t^2 - 3t)^2}$$

Для выполнения дифференцирования применяется также дифференциальный оператор **D**[i](f), где f — выражение, задающее функцию, i — натуральное число. Если f — функция от одного аргумента, то **D**(f) вычисляет производную от f, например

```
> D(sin);
```

$$\cos$$

Производная также функция одного аргумента **D**(f)(x) = **diff**(f(x), x). Таким образом **D**(f) эквивалентно **unapply**(**diff**(f(x), x), x).

```
> D(sin)(x);
```

$$\cos(x)$$

```
> D(sin)(Pi);
```

$$-1$$

Если f — функция n аргументов, то **D**[i](f) вычисляет частную производную по отношению к i-тому аргументу. В общем случае **D**[i,j](f) эквивалентно **D**[i](**D**[j](f)) и **D**[](f) = f.

Приведем примеры.

```
> D(exp+cos^2+Pi+tan);
```

$$\exp - 2 \sin \cos + 1 + \tan^2$$

```
> D(ln);
```

$$A \to 1/A$$

```
> D(D(f));
```

$$(D^{(2)})(f)$$

Для многократного дифференцирования функции от одной переменной применяется оператор D@@n, где n — кратность дифференцирования.

```
> (D@@2)(f);
```

$$(D^{(2)})(f)$$

```
> (D@@n)(f);
```

$$(D^{(n)})(f)$$

Производная от композиции двух функций (сложной функции):

```
> D(f@g);
```

$$(D(f))@g\ \mathrm{D}(g)$$

```
> D(sin@y);
```

$$\cos@y\ \mathrm{D}y$$

Для выполнения многократного дифференцирования по заданной переменной функции от нескольких переменных применяется оператор **D**[i$n](f), где i — номер переменной, n — кратность дифференцирования.

```
> D[i$n](f);
```

$$\mathrm{D}_{i\,\$\,n}(f)$$

```
> D[i,j](f);
```

$$\mathrm{D}_{i,j}(f)$$

```
> D[i,j](f)-D[j,i](f);
```

$$0$$

```
> D[i](D[j,i](f));
```

$$D_i(D_{i,j}(f))$$

Оператор дифференцирования применяется также к функциональным операторам и процедурам.

```
> f := x -> x^2;
```

$$f := x \to x^2$$

```
> D(f);
```

$$x \to 2\,x$$

```
> f := (x,y) -> exp(x*y);
```

$$f := (x, y) \to \mathbf{e}^{(xy)}$$

```
> D[](f);
```

$$f$$

```
> D[1](f);
```

$$(x, y) \to y\,\mathbf{e}^{(xy)}$$

```
> D[2](f);
```

$$(x, y) \to y\,\mathbf{e}^{(xy)}$$

Пусть

```
> f := proc(x) local t1,t2;
  t1 := x^2;
  t2 := sin(x);
  3*t1*t2+2*x*t1-x*t2
  end:
```

Вычислим производную от f по аргументу x.

```
> D(f);
```

```
proc(x)
local t2, t1x, t2x, t1;
    t1x := 2*x;
    t1 := x^2;
    t2x := cos(x);
    t2 := sin(x);
    3*t2*t1x + 3*t1*t2x + 2*t1 + 2*x*t1x - t2 - x*t2x
end
```

Проверим правильность вычисления производной.

```
> D(f)(x) - diff(f(x),x);
```

$$0$$

6.5. Пределы

Для нахождения предела выражения или функции в *Maple* используется команда **limit**(параметр1, параметр2). Первый параметр — выражение, второй параметр — имя переменной, приравненное значению переменной в точке предела. Необязательный третий параметр — направление предела. Если направление не задано, вычисляется стандартный двусторонний предел. Если предел не существует, в качестве ответа возвращается сообщение "undefined". Если *Maple* не способен вычислить предел (однако он может существовать), возвращается невыполненная команда.

```
> limit( cos(x)/x, x=Pi/2 );
```

$$0$$

```
> limit(( -x^2+x+1)/ (x+4), x=infinity );
```

$$-\infty$$

```
> limit( tan(x), x=Pi/2);
```

$$undefined$$

В большом количестве случаев выражение, которое не имеет двустороннего предела, имеет односторонний предел:

```
> limit(  tan(x), x=Pi/2, left);
```

$$\infty$$

```
> limit(  tan(x), x=Pi/2, right);
```

$$-\infty$$

```
> limit((1+a/x)^x,x=infinity);
```

$$e^a$$

В команде **limit** может присутствовать также необязательная опция **complex** или **real** в качестве третьего параметра аргумента. Эта опция определяет, в комплексной или действительной области вычисляется предел.

6.6. Интегрирование

Как определенные, так и неопределенные интегралы в программе *Maple* вычисляются при помощи команды **int**(параметр1, параметр2).

Аналитическое интегрирование

Сначала рассмотрим неопределенное интегрирование. В этом случае первым параметром аргумента команды **int** является интегрируемое выражение, а вторым — переменная интегрирования. Результат интегрирования (если он найден) выводится без стандартной константы интегрирования. Этим достигается возможность многократного использования результата в дальнейших вычислениях. Если **int** не находит интеграла, команда возвращается невыполненной. Далее — некоторые примеры неопределенных интегралов.

```
> int(2*x*exp(x^2),x);
```

$$e^{(x^2)}$$

В определенных интегралах ко второму параметру команды **int** добавляются пределы интегрирования.

```
> restart;
  int(sin(a*x)^2/(x^2),x=0..infinity);
```

$$\frac{1}{2}\,\mathrm{signum}(a)\,\pi\,a$$

Рассмотрим еще один интеграл

```
> int(log(x)^a,x=0..1);
Definite integration: Can't determine if the integral is convergent.
Need to know the sign of --> 1+a
Will now try indefinite integration and then take limits.
```

$$\int_0^1 \ln(x)^a\,dx$$

Maple уточняет знак величины 1+a. Предположим, что a>−1:

```
> assume(a>-1);int(log(x)^a,x=0..1);
```

$$(-1)^{a\sim}\,\Gamma(A)\,A$$

Теперь все в порядке. Вообще, команда **assume** во многих случаях позволяет не только найти значение интеграла, не имеющего в общем случае первообразной, но и значительно упростить полученное выражение, как, например, в следующем примере

```
> int(1/(x^2+a^2)/(x^2+b^2),x=0..infinity);
```

$$\frac{1}{2}\,\frac{\pi\,(-\mathrm{csgn}(\bar{a})\,b+\mathrm{csgn}(\bar{b})\,a)}{(a^2-b^2)\,a\,b}$$

Введем предположения относительно параметров

```
> assume(a>0,b>0);
  int(1/(x^2+a^2)/(x^2+b^2),x=0..infinity);
```

$$\frac{1}{2}\,\frac{\pi}{b{\sim}\,a{\sim}\,(b{\sim}+a{\sim})}$$

Опция **continuous**, добавленная в качестве четвертого аргумента команды **int**, вынуждает *Maple* игнорировать любые возможные разрывы подынтегральной функции в диапазоне интегрирования.

```
> int( 1/(x-1)^2, x=0..2,continuous);
```

$$-2$$

В отличие от команды **diff** простым добавлением переменных интегрирования в команду **int** невозможно задать многократное интегрирование. С этой целью интеграл по одной из переменных включается в качестве первого параметра в аргумент интеграла по другой переменной:

```
> int ( int ( x^2 * y^3, x ), y);
```

$$\frac{1}{12}x^3y^3$$

```
> int ( int ( int ( x^2 *y^2* z^2, x=1 ..2 ),
  y=1 ..2 ), z=1 ..2 );
```

$$\frac{343}{27}$$

В следующем примере интеграл выражается через полный эллиптический интеграл первого рода **EllipticK**, для упрощения сложных радикалов применяется команда **radnormal**.

```
> Int( 1/sqrt( sin(x) ),
  x=0..Pi/2 )= int( 1/sqrt( sin(x) ),
  x=0..Pi/2 ):radnormal(");
```

$$\int_0^{1/2\,\pi} \frac{1}{\sqrt{\sin(x)}}\,dx = -4\,\frac{\mathrm{EllipticK}(3-2\sqrt{2})\,(3\sqrt{2}-4)}{-2+\sqrt{2}}$$

Численное интегрирование

Для выполнения численного интегрирования используется команда **evalf**.

```
> int (1/(exp (x^2)+x), x=0..1);
```

$$\int_0^1 \frac{1}{e^{(x2)}+x}\,dx$$

```
> evalf ( int ( 1/( exp (x^2)+x), x=0 ..1 ));
```

$$.5859203128$$

В следующем примере при помощи команды **evalf**(int, 20) находим приближенное значение интеграла с точностью 20 значащих цифр.

```
> int( exp(v-v^2/2)/(1+1/2*exp(v)),
  v = -infinity..infinity );

> evalf(", 20);
```

$$\int_{-\infty}^{\infty} \frac{e^{(v-\frac{1}{2}\,v^2)}}{1+\frac{1}{2}\,e^v}\,dx$$

$$1.8055770622970496788$$

6.7. Суммы и произведения

Конечные и бесконечные суммы вычисляются командой **sum**(параметр1, параметр2). Параметрами, также как в команде интегрирования, являются выражение, переменная и пределы суммирования. В конечных суммах диапазон суммирования может содержать числовые или символьные значения.

Приведем несколько примеров.

```
> sum( 'i^2', 'i'=1..100 );
```

$$338350$$

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

$$\frac{1}{3}(n+1)^3 - \frac{1}{2}(n+1)^2 + \frac{1}{6}n + \frac{1}{6}$$

```
> sum( '2^i/2 * i', 'i'=a...d);
```

$$\frac{1}{2}2^{(d+1)}(d+1) - 2^{(d+1)} - \frac{1}{2}2^a a + 2^a$$

Кавычки обязательны в индексах суммирования.

Для задания бесконечного суммирования в качестве правой границы диапазона переменной суммирования устанавливается значение **infinity**.

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

$$\frac{1}{6}\pi^2$$

```
> sum((-1)^n/(3*n+1)^3,n=0..infinity);
```

$$\mathrm{hypergeom}\left(\left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, 1\right], \left[\frac{4}{3}, \frac{4}{3}, \frac{4}{3}\right], -1\right)$$

```
> evalf(");
```

$$.9865909863$$

Если *Maple* не способен найти аналитическую форму для некоторого интеграла или суммы, он просто повторяет команду интегрирования (суммирования) в выводе. Приведем еще пример бесконечного произведения

```
> product(a^(n/(n+1)^2)/sqrt((1+log(a)/n)*(1+log-
   (a)/ (n+1/2))),n=1..infinity);
```

$$\prod_{n=1}^{\infty} \frac{a^{\left(\frac{n}{(n+1)^2}\right)}}{\sqrt{\left(1+\frac{\ln(a)}{n}\right)\left(1+\frac{\ln(a)}{n+\frac{1}{2}}\right)}}$$

При произвольном значении *a Maple* не находит замкнутой формы этого произведения, но положим a=7

```
> a:=7;product(a^(n/(n+1)^2)/sqrt((1+log(a)/n)*(1+
   log(a)/(n+1/2))),n=1..infinity);
```

$$a = 7$$

$$\frac{\sqrt{\Gamma\,(\ln(7))\,\ln(7)}\sqrt{\frac{1}{2}\,\Gamma\left(\frac{1}{2}+\ln(7)\right)+\Gamma\left(\frac{1}{2}+\ln(7)\right)\ln 7}\,\sqrt{2}\,7^{\gamma}}{\pi^{1/4}\,\left(7^{(\pi^2)}\right)^{1/6}}$$

```
> simplifi (");
```

$$7^{(-1/6\,\pi^2+\gamma)}\,4^{(-1/2\,\ln(7))}\,\sqrt{\Gamma(2+2\ln(7))}$$

```
> evalf(");
```

$$.3248180834$$

6.8. Примеры из линейной алгебры

В этом разделе мы кратко остановимся на возможностях системы *Maple* для решения задач линейной алгебры. Начнем с рассмотрения типов объектов, используемых в линейной алгебре, а затем сделаем краткий обзор команд, доступных в пакете **linalg**.

Массивы

Один из типов объектов линейной алгебры — массив. Массив — структура высокого уровня (одной или больше размерностей), содержащая совокупность индивидуальных объектов. Каждой размерности массива соответствует диапазон,

определяемый пользователем. Массив одной размерности — это список, двух размерностей — это список списков. Вектор эквивалентен одномерному массиву с нижней границей, равной 1, матрица эквивалентна двумерному массиву с обеими нижними границами, равными 1. Массивы создаются при помощи команды **array**. Индивидуальные элементы массива могут или определяться заранее, или оставаться неизвестными.

Приведем пример массива 3 × 3, который содержит заранее определенные элементы.

```
> array ( 1 ..3, 1 ..3, [[a,b,c ], [d,e,f ],
  [g,h,i ] ] );
```

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Как видите, *Maple* распечатывает массив в виде матрицы. В пакете **linalg** имеются команды, которые позволяют создавать матрицу или вектор непосредственно, но они — только специальные случаи команды **array**.

Специальные типы матриц

Имеется небольшое количество специальных типов матриц, которые могут автоматически генерироваться в *Maple*. Это — симметричная, антисимметричная, разреженная, диагональная и единичная матрицы. Такие матрицы создаются командой **array** с добавлением типа матрицы как дополнительного параметра.

```
> array ( 1 ..2, 1 ..2, identity);
```

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

```
> array ( 1 ..2, 1 ..6, [(1,3)=p,(2,4)=r], sparse );
```

$$\begin{bmatrix} 0 & 0 & p & 0 & 0 & 0 \\ 0 & 0 & 0 & r & 0 & 0 \end{bmatrix}$$

Управление элементами массивов

Есть некоторые отличия в правилах оперирования программы *Maple* с массивами по сравнению с менее сложными объектами. Если Вы присвоили массиву имя переменной, а затем захотите вывести значение этой переменной, просто записав ее в виде команды, то результатом будет не сам массив, а имя переменной, которое ему присвоено. Это правило называется "lust name evaluation" (вычисление до последнего имени). Для вывода самого массива можно использовать одну из команд **op** или **evalm** (для матриц).

```
> M:= array(1..2,1..2, [[3,4], [6,5]]);
```

$$M := \begin{bmatrix} 3 & 4 \\ 6 & 5 \end{bmatrix}$$

```
> M;op(M);
```

$$M$$

$$\begin{bmatrix} 3 & 4 \\ 6 & 5 \end{bmatrix}$$

```
> evalm(M);
```

$$\begin{bmatrix} 3 & 4 \\ 6 & 5 \end{bmatrix}$$

```
> M:= 'M';
```

$$M := M$$

Правило вычисления до последнего имени также используется для таких сложных объектов, как таблицы и процедуры *Maple*. Это правило применяется с целью избежать громоздких выводов на экран дисплея. На индивидуальные элементы массивов можно ссылаться указанием номера строки и номера столбца, в которых находится элемент. Заметим однако, что для элементов массива правило вычисления до последнего имени не работает. Можно также переопределять элементы массива после того, как массив уже создан — присвоением новых значений их именам.
Приведем пример.

```
> M:= array( 1 ..2, 1 ..3, [[2,8,32], [45,-1,0] ] );
```

$$M := \begin{bmatrix} 2 & 8 & 32 \\ 45 & -1 & 0 \end{bmatrix}$$

```
> M[1,3];
```

$$32$$

```
> M[2,3 ]:= x-3;op(M);
```

$$M_{2,3} := x - 3$$

$$\begin{bmatrix} 2 & 8 & 32 \\ 45 & -1 & x-3 \end{bmatrix}$$

```
> M:= 'M';
```

$$M := M$$

Команды пакета linalg

Пакет линейной алгебры (**linalg**) включает много полезных команд. Указатель каталога всех этих команд загрузим при помощи команды **with**.

```
> with(linalg):
Warning, new definition for norm
Warning, new definition for norm
```

Теперь все команды пакета доступны для непосредственного использования. Сначала зададим две матрицы с которыми будем работать.

```
> A:= array( 1 ..3, 1 ..3, [[3,-1,1 ], [-1,5,-1 ],
  [1,-1,3 ] ] );
```

$$A := \begin{bmatrix} 3 & -1 & 1 \\ -1 & 5 & -1 \\ 1 & -1 & 3 \end{bmatrix}$$

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

$$B := \begin{bmatrix} y & y & -1 \\ -y & y & 0 \\ y & 0 & 0 \end{bmatrix}$$

Матрицы можно складывать

```
> evalm(A + B);
```

$$\begin{bmatrix} 3+y & -1+y & 0 \\ -1-y & 5+y & -1 \\ 1+y & -1 & 3 \end{bmatrix}$$

и перемножать

```
> evalm(A&*B);multiply(A,B);
```

$$\begin{bmatrix} 5y & 2y & -3 \\ -7y & 4y & 1 \\ 5y & 0 & -1 \end{bmatrix}$$

$$\begin{bmatrix} 5y & 2y & -3 \\ -7y & 4y & 1 \\ 5y & 0 & -1 \end{bmatrix}$$

транспонировать

```
> transpose (A);
```

$$\begin{bmatrix} 3 & -1 & 1 \\ -1 & 5 & -1 \\ 1 & -1 & 3 \end{bmatrix}$$

обращать:

```
> inverse (A);
```

$$\begin{bmatrix} \frac{7}{18} & \frac{1}{18} & \frac{-1}{9} \\ \frac{1}{18} & \frac{2}{9} & \frac{1}{18} \\ \frac{-1}{9} & \frac{1}{18} & \frac{7}{18} \end{bmatrix}$$

можно найти детерминант матрицы:

```
> det (B);
```

$$y^2$$

определить собственные значения и собственные векторы

```
> eigenvals(A);
```

$$6,\ 2,\ 3$$

```
> v := eigenvects(A);
```

$$v := [2, 1, \{[-1, 0, 1]\}], [3, 1, \{[1, 1, 1]\}], [6, 1, \{[1, -2, 1]\}]$$

В приведенной записи указываются собственные значения (характеристические числа), их кратность и в фигурных скобках соответствующие им собственные вектора.

В *Maple* используется оператор **&*** для некоммутативного перемножения матриц.

```
> evalm(B&*B);
```

$$\begin{bmatrix} -y & 2y^2 & -y \\ -2y^2 & 0 & y \\ y^2 & y^2 & -y \end{bmatrix}$$

Для умножения матрицы на себя можно использовать также оператор возведения в степень.

```
> evalm(B^2);
```

$$\begin{bmatrix} -y & 2y^2 & -y \\ -2y^2 & 0 & y \\ y^2 & y^2 & -y \end{bmatrix}$$

Maple "знает" также матричные экспоненты $\mathbf{e}^A$, которые, например, используются в теории систем обыкновенных дифференциальных уравнений

```
> exponential(A);
```

$$\begin{bmatrix} \frac{1}{3}\mathbf{e}^3 + \frac{1}{6}\mathbf{e}^6 + \frac{1}{2}\mathbf{e}^2 & \%1 & \frac{1}{3}\mathbf{e}^3 + \frac{1}{6}\mathbf{e}^2 + \frac{1}{2}\mathbf{e}^6 \\ \%1 & \frac{1}{3}\mathbf{e}^6 + \frac{1}{6}\mathbf{e}^3 & \%1 \\ \frac{1}{3}\mathbf{e}^3 + \frac{1}{6}\mathbf{e}^2 + \frac{1}{2}\mathbf{e}^6 & \%1 & \frac{1}{3}\mathbf{e}^3 + \frac{1}{6}\mathbf{e}^6 + \frac{1}{2}\mathbf{e}^2 \end{bmatrix}$$

$$\%1 := -\frac{1}{3}\mathbf{e}^6 + \frac{1}{6}\mathbf{e}^3$$

Другие команды пакета **linalg** осуществляют преобразования матриц. Выделение субматрицы:

```
> subm:= submatrix( B, 1 ..2, 2 ..3 );
```

$$subm := \begin{bmatrix} y & -1 \\ y & 0 \end{bmatrix}$$

Присоединение одной матрицы слева от другой:

```
> augment( A, B);
```

$$\begin{bmatrix} 3 & -1 & 1 & y & y & -1 \\ -1 & 5 & -1 & -y & y & 0 \\ 1 & -1 & 3 & y & 0 & 0 \end{bmatrix}$$

Вертикальное объединение матриц:

```
> linalg[stack]( A, B);
```

$$\begin{bmatrix} 3 & -1 & 1 \\ -1 & 5 & -1 \\ 1 & -1 & 3 \\ y & y & -1 \\ -y & y & 0 \\ y & 0 & 0 \end{bmatrix}$$

Maple "знает" некоторые специальные типы матриц: Фибоначчи, Вандермонда, Гильберта, Теплица.

Приведем пример

```
> V := vandermonde( [x.(1..3)] );
```

$$\begin{bmatrix} 1 & x1 & x1^2 \\ 1 & x1 & x1^2 \\ 1 & x1 & x1^2 \end{bmatrix}$$

```
> factor(det(V));
```

$$(x2 - x1)\,(-x1 + x3)\,(x3 - x2)$$

6.9. Обыкновенные дифференциальные уравнения

Для решения обыкновенных дифференциальных уравнений и систем используется команда **dsolve**(уравнения, переменные, опции), где уравнения — заданные дифференциальные уравнения, переменные — те переменные, по отношению к которым ищется решение, опции — необязательные опции, задаваемые в виде: ключевое слово=значение.

Команда **dsolve** способна решать аналитически большое количество дифференциальных уравнений.

Если задана опция **type=exact**, то команда пытается найти аналитическое решение — эта опция задана по умолчанию. Другие возможные значения этой опции **type=series** (в этом случае решение ищется в виде ряда) и **type=numeric** (в этом случае ищется численное решение).

Можно задать еще несколько опций, которые определяют: будет ли решение искаться в явном виде или нет (**explicit=true** или **explicit=false**), задают метод решения (например **method=laplace**).

Если задана опция **type=numeric**, то можно также задать метод численного расчета, например

- **method=rkf45** — метод Рунге-Кутта четвертого-пятого порядка,
- **method=dverk78** — метод Рунге-Кутта седьмого-восьмого порядка,
- **method=classical** — содержит несколько классических методов (Эйлера, Рунге-Кутта третьего порядка и некоторые другие),
- **method=gear и method=mgear** — одношаговый и многошаговый методы Гира.

Приведем примеры

```
> dsolve(diff(y(x),x$2) - y(x) = sin(x)*x, y(x));
```

$$y(x) = -\frac{1}{2}\cos(x) - \frac{1}{2}\sin(x)\ x + _C1\ e^x + _C2\ e^{(-x)}$$

В полученном решении _C1, _C2 — произвольные постоянные. Начальные условия в дифференциальных уравнениях задаются через запятую, при этом уравнение и начальное условие объединяются фигурными скобками в набор.

```
> dsolve({diff(v(t),t)+2*t=0, v(1)=5}, v(t));
```

$$v(t) = -t^2 + 6$$

Производные в начальных условиях записываются в операторном виде как D(D(y))(0) или (D@@2)(y)(0)).

```
> de1 := diff(y(t),t$2) + 5*diff(y(t),t) + 6*y(t) = 0;
```

$$de1 := \left(\frac{\partial^2}{\partial t^2}y(t)\right) + 5\left(\frac{\partial}{\partial t}y(t)\right) + 6\ y(t) = 0$$

```
> dsolve({de1, y(0)=0, D(y)(0)=1}, y(t), method=
  laplace);
```

$$y(t) = -e^{(-3t)} + e^{(-2t)}$$

Уравнение четвертого порядка

```
> de2 := diff(y(x),x$4)+2*diff(y(x),x$2) -cos(x)=3;
```

$$de2 := \left(\frac{\partial^4}{\partial x^4}y(x)\right) + 2\left(\frac{\partial}{\partial x^2}y(x)\right) - \cos(x) = 3$$

```
> dsolve(de2,y(x)):combine(");
```

$$y(x) = -\cos(x) + \frac{3}{4}x^2 - \frac{3}{4} + _C1 + _C2\,x + _C3\cos(\sqrt{2}\,x) + _C4\sin(\sqrt{2}\,x)$$

Для следующего уравнения решение находится только методом замены переменных

```
> q:=(2*sqrt(x*y(x))-x)*diff(y(x),x)+y(x);
```

$$q := (2\sqrt{x\,y(x)} - x)\left(\frac{\partial}{\partial x}y(x)\right) + y(x)$$

Для замены переменных применяется команда **Dchangevar** пакета **DEtools**

```
> with(DEtools):f:=Dchangevar({y(x)=v(x)*x},
  [q],x);
```

$$f := (2\sqrt{x^2\,v(x)} - x)\left(\left(\frac{\partial}{\partial x}v(x)\right)x + v(x)\right) + v(x)\,x$$

Теперь находим решение для функции v(x)

```
> w:=dsolve(f,v(x));
```

$$w := x = \frac{_C1\,x\,\mathbf{e}^{\left(-\frac{x}{\sqrt{x\ v(x)}}\right)}}{v(x)}$$

Чтобы получить искомое решение y(x), делаем обратную подстановку

```
> subs({v(x)=y(x)/x},w);
```

$$w := x = \frac{_C1\,x\,\mathbf{e}^{\left(-\frac{x}{\sqrt{x\,y(x)}}\right)}}{y(x)}$$

```
> y(x)=solve(",y(x));
```

$$y(x) = \left(\frac{1}{4}\frac{x}{\mathrm{LambertW}\left(\frac{1}{2}\frac{1}{\sqrt{\frac{_C1}{x}}}\right)^2}, \frac{1}{4}\frac{x}{\mathrm{LambertW}\left(\frac{1}{2}\frac{1}{\sqrt{\frac{_C1}{x}}}\right)^2}\right)$$

Команда **dsolve** позволяет также находить базисный набор функций, линейная комбинация которых даст полное решение дифференциального уравнения. Для этого в команду **dsolve** добавляется опция **output=basis**:

```
> dsolve(2*x*diff(y(x),x$2)+diff(y(x),x)+3*y(x)=x,
  y(x),output=basis);
```

$$\left[\left[\frac{x^{1/4}\sin(\sqrt{6}\,\sqrt{x}\,)}{\sqrt{\sqrt{6}\,\sqrt{x}}}, \frac{x^{1/4}\cos(\sqrt{6}\,\sqrt{x}\,)}{\sqrt{\sqrt{6}\,\sqrt{x}}}\right], -\frac{1}{9}+\frac{1}{3}x\right]$$

Система дифференциальных уравнений вместе с начальными данными записывается в виде набора (последовательности выражений в фигурных скобках) в аргументе команды **dsolve**

```
> sys := diff(y(x),x)=z(x)-y(x)-x, diff(z(x),x)=y(x):
> fcns := {y(x), z(x)}:
> dsolve({sys,y(0)=0,z(0)=1}, fcns);
```

$$\{z(x) = -\frac{1}{5}\sqrt{5}\,e^{(1/2\,(\sqrt{5}-1)\,x)} + \frac{1}{5}\sqrt{5}\,e^{(-1/2\,(\sqrt{5}+1)\,x)} + x + 1,$$

$$y(x) = \frac{1}{10}\sqrt{5}\,e^{(1/2\,(\sqrt{5}-1)\,x)} - \frac{1}{10}\sqrt{5}\,e^{(-1/2\,(\sqrt{5}+1)\,x)} + 1 - \frac{1}{2}\,e^{(-1/2\,(\sqrt{5}+1)\,x)} -$$

$$-\frac{1}{2}\,e^{(1/2\,(\sqrt{5}-1)\,x)}\}$$

Решение этой же системы можно найти также в виде степенных рядов

```
> dsolve({sys,y(0)=0,z(0)=1}, fcns, type=series);
```

$$\{z(x) = 1 + \frac{1}{2}x^2 - \frac{1}{3}x^3 + \frac{1}{8}x^4 - \frac{1}{24}x^5 + O(x^6),$$

$$y(x) = x - x^2 + \frac{1}{2}x^3 - \frac{5}{24}x^4 + \frac{1}{15}x^5 + O(x^6)\}$$

Численное решение той же системы достигается простой установкой опции **type=numeric**

```
> F := dsolve({sys,y(0)=0,z(0)=1}, fcns, type=
  numeric);
```

$$F := \textbf{proc}(rkf45_x)\ \dots\ \textbf{end}$$

Как видим, решение выводится в виде процедуры нахождения численных значений методом Рунге-Кутта, используемым программой по умолчанию. Чтобы найти решение при значении независимой переменной x, равном, скажем, 1, достаточно записать:

```
> F(1);
```

$$[x = 1, z(x) = 1.258972076536823, y(x) = .3437314082767540]$$

В следующем примере в команде **dsolve** явно указан метод решения системы дифференциальных уравнений, а также массив начальных значений независимой переменной, для которых мы хотим получить результат.

```
> sys2 := {(D@@2)(y)(x)=2*x^3*y(x), y(0)=1,
  D(y)(0)=1}:
  s := dsolve(sys2, {y(x)}, type=numeric,
  method=dverk78, value=array([1.0,1.5,1.7]));
```

$$s := \left[\begin{array}{c} \left[x,\ y(x),\ \frac{\partial}{\partial x}\,y(x)\right] \\ \left[\begin{array}{ccc} 1. & 2.170132435253142 & 1.936037883117915 \\ 1.500000000000000 & 4.268267966270414 & 8.363916916540699 \\ 1.700000000000000 & 6.710398546651994 & 17.27579721228745 \end{array}\right] \end{array}\right]$$

Так можно извлечь из массива конкретное значение.

```
> s[2,1][2,3];
```

$$8.3639169165406990 2$$

Программа содержит несколько алгоритмов для численного решения жестких уравнений и систем дифференциальных уравнений. Один из таких алгоритмов — метод одношаговой интерполяции Гира применен для решения следующего уравнения

```
> Digits := 10:
  deq1 := {diff(y(x), x$3) = y(x)*diff(y(x), x)-x}:
  init1 := {(D@@2)(y)(1) = 4, D(y)(1) = 3,
  y(1) = 2.4 }:
  ans1 := dsolve(deq1 union init1, y(x), type=numeric,
  method=gear[polyextr], stepsize=0.015,
  minstep=Float(1,-11), errorper=Float(1,-5)):
  ans1(1.01);
```

$$\left[x = 1.01,\ \mathrm{y}(x) = 2.430201040709296,\quad \frac{\partial}{\partial x}\,\mathrm{y}(x) = 3.040312954671420,\right.$$

$$\left.\frac{\partial^2}{\partial x^2}\,\mathrm{y}(x) = 4.062888549132273\right]$$

При решении следующего дифференциального уравнения применен метод многошаговой интерполяции Гира.

```
> Digits := 12:
deq3 := diff(y(t), t$2) = 100*(exp(-10*t)+
exp (10*t)):
ans3 := dsolve({deq3}, y(t), numeric, method=
mgear [msteppart], initial=array([2,0]), start=0):
ans3(0.7653);
```

$$\left[t = .7653,\ \mathrm{y}(t) = 2106.958269487619,\quad \frac{\partial}{\partial t}\,\mathrm{y}(t) = 21069.56916506877\right]$$

Рассмотрим дифференциальное уравнение Ньютона, описывающее движение тела под действием периодической силы a*cos(omega*(x–x0)) и учитывающее упругую силу kf(x)

```
> restart;
deq:= m*diff(f(x),x,x)+k*f(x)=a*cos(omega*(x-x0));
```

$$deq := m\left(\frac{\partial^2}{\partial x^2}\,\mathrm{f}(x)\right) + k\mathrm{f}(x) = a\,\cos(\mathrm{w}\,(x - x0))$$

Для решения этого уравнения применим метод преобразования Фурье

```
> with(inttrans):F:=fourier(deq,x,xi);
```

$$F := -m\,\xi^2\,\mathrm{fourie}(\mathrm{f}(x), x, \xi) + k\,\mathrm{fourie}(\mathrm{f}(x), x, \xi) =$$

$$= \mathrm{a}\left(\frac{1}{2}\,\mathbf{e}^{(-I\,\omega\,x0)}\,\mathrm{fourie}(\mathbf{e}^{(I\,\omega\,x)}, x, \xi) + \frac{1}{2}\,\mathbf{e}^{(I\,\omega\,x0)}\,\mathrm{fourie}(\mathbf{e}^{(-I\,\omega\,x)}, x, \xi)\right)$$

Теперь решим полученное алгебраическое уравнение относительно преобразования Фурье от искомой функции f(x)

```
> S:=solve(F,fourier(f(x),x,xi));
```

$$S := \frac{1}{2}\,\frac{a\,(e^{(-I\,\omega\,x0)}\,\mathrm{fourie}(e^{(I\,\omega\,x)}, x, \xi) + e^{(I\,\omega\,x0)}\,\mathrm{fourie}(e^{(-I\,\omega\,x)}, x, \xi))}{-m\,\xi^2 + k}$$

При разумном допущении относительно частоты колебаний ω получим после упрощения

```
> assume(omega>0);simplify(S);
```

$$\frac{a\,\pi\,(e^{(-I\,\omega\sim\,x0)}\,\mathrm{Dirac}(\xi - \omega\sim) + e^{(I\,\omega\sim\,x0)}\,\mathrm{Dirac}(\xi - \omega\sim))}{-m\,\xi^2 + k}$$

В последнюю формулу входит обобщенная функция Дирака **Dirac**. Теперь при помощи обратного Фурье–преобразования найдем искомую функцию

```
> invfourier(",xi,x);
```

$$\frac{1}{2}\,\frac{a\,(e^{(-I\,\omega\sim\,x0)}\,(e^{(I\,\omega\sim\,x)} + e^{(I\,\omega\sim\,x0)}\,(e^{(-I\,\omega\sim\,x)})}{-m\,\omega\sim^2 - k}$$

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

$$-\frac{a\cos(\omega\sim\,(x - x0))}{-m\,\omega\sim^2 + k}$$

6.10. Уравнения в частных производных

Новая версия программы *Maple* “способна” решать аналитически некоторые классы дифференциальных уравнений в частных производных. Для этого введена команда **pdesolve**(уравнения, переменные).

Приведем примеры.

```
> restart;pdesolve(diff(f(x,y),x,x)+5*diff(f(x,y),
  x,y)=3, f(x,y));
```

$$f(x, y) = \frac{3}{2}\,x^2 + _F1(y) + _F2(y - 5\,x)$$

В решении этого уравнения присутствуют произвольные функции _F1, _F2.

```
> pdesolve( 3*diff(g(x,y),x)+7*diff(g(x,y),x,y)=x*y,
  g(x,y) );
```

$$g(x, y) = \frac{1}{6}\,x^2\,y - \frac{7}{18}\,x^2 + _F1(y) + e^{(-3/7\,y)}\,_F2(x)$$

Maple находит решение некоторых видов уравнений с переменными коэффициентами, например

```
> pdesolve(y*diff(U(x,y),x)+x*diff(U(x,y),y)=0,
  U(x,y));
```

$$U(x, y) = _F1(-x^2 + y^2)$$

Неоднородное уравнение для функции U от трех независимых переменных

```
> pdesolve( diff(U(x, y, z), x)+2*diff(U(x, y, z),
  y)+5*diff(U(x, y, z), z)=13*x*y*z, U(x, y, z) );
```

$$U(x, y, z) = \frac{65}{6}x^4 - \frac{65}{6}x^3y - \frac{13}{3}x^3z + \frac{13}{2}x^2\,y\,z + _F1(y - 2\,x, z - 5\,x)$$

Следующие два примера уравнений математической физики.
Уравнение теплопроводности

```
> restart;heat:=diff(u(x,t),t)-k*diff(u(x,t),x,x)=0;
```

$$heat := \left(\frac{\partial}{\partial t}\,u(x, t)\right) - k\left(\frac{\partial^2}{\partial x^2}\,u(x, t)\right) = 0$$

Команда **pdesolve** “в лоб” не решает это уравнение, действительно

```
> pdesolve(heat,u(x,t));
```

$$\mathrm{pdesolve}\left(\left(\frac{\partial}{\partial t}\,u(x, t)\right) - k\left(\frac{\partial^2}{\partial x^2}\,u(x, t)\right) = 0, u(x, t)\right)$$

Применим известный прием разделения переменных. Для этого вначале сделаем подстановку

```
> eq:=subs(u(x,t)=X(x)*T(t),heat);
```

$$eq := \left(\frac{\partial}{\partial t}\,X(x)\,T(t)\right) - k\left(\frac{\partial^2}{\partial x^2}\,X(x)\,T(t)\right) = 0$$

Теперь разделим обе части уравнения на X(x)*T(t)

```
> expand(eq/X(x)/T(t));
```

$$\frac{\frac{\partial}{\partial t}\mathrm{T}(t)}{\mathrm{T}(t)} - \frac{\left(k\,\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)}{\mathrm{X}(x)} = 0$$

Разделим переменные

```
> sep:=(")+(k*diff(X(x),x,x)/X(x)=k*diff(X(x),x,x)/
   X(x));
```

$$sep := \frac{\frac{\partial}{\partial t}\mathrm{T}(t)}{\mathrm{T}(t)} = \frac{\left(k\,\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)}{\mathrm{X}(x)}$$

Так как в левой и правой частях полученного равенства стоят функции от разных переменных, то правая и левая части являются постоянными величинами.

```
> lhs(sep)=C;
```

$$\frac{\frac{\partial}{\partial t}\mathrm{T}(t)}{\mathrm{T}(t)} = C$$

Теперь мы получили обыкновенное дифференциальное уравнение и его решение

```
> T_sol:=dsolve(",T(t));
```

$$T_sol := \mathrm{T}(t) = \mathbf{e}^{(C\,t)}\,_C1$$

Аналогично приравняем константе правую часть равенства

```
> rhs(sep)=C;
```

$$\frac{k\left(\frac{\partial^2}{\partial x^2}\mathrm{X}(x)\right)}{\mathrm{X}(x)} = C$$

Решение полученного обыкновенного дифференциального уравнения

```
> X_sol:=dsolve(",X(x),explicit=true);
```

$$X_sol := \mathrm{X}(x) = -\frac{1}{2}\,\frac{_C1\,k^2 - \left(\mathbf{e}^{\left(-\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\right)^2}{\mathbf{e}^{\left(-\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\sqrt{kC}},$$

$$\mathrm{X}(x) = -\frac{1}{2}\,\frac{_C1\,k^2 - \left(\mathbf{e}^{\left(\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\right)^2}{\mathbf{e}^{\left(\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\sqrt{kC}}$$

```
> map(subs,[X_sol],T_sol,X(x)*T(t));
```

$$\left[-\frac{1}{2}\,\frac{\left(_C1\,k^2 - \left(\mathbf{e}^{\left(-\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\right)^2\right)\mathbf{e}^{(C\,t)}\,_C1}{\mathbf{e}^{\left(-\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\sqrt{kC}},\right.$$

$$\left.-\frac{1}{2}\,\frac{\left(_C1\,k^2 - \left(\mathbf{e}^{\left(\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\right)^2\right)\mathbf{e}^{(C\,t)}\,_C1}{\mathbf{e}^{\left(\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\sqrt{kC}}\right]$$

```
> sol:=map(simplify,");
```

$$sol := \left[\frac{1}{2}\,\frac{\left(-\,_C1\,k^2 + \mathbf{e}^{\left(-2\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\right)_C1\,\mathbf{e}^{\left(\frac{x\sqrt{kC}\;+\;_C2\sqrt{kC}+C\,t\,k}{k}\right)}}{\sqrt{kC}},\right.$$

$$\left.\frac{1}{2}\frac{\left(-_C1\,k^2+e^{\left(2\frac{\sqrt{kC}\,(x+_C2)}{k}\right)}\right)_C1\,e^{\left(-\frac{-C\,t\,k+x\sqrt{kC}+_C2\sqrt{kC}}{k}\right)}}{\sqrt{kC}}\right]$$

Для упрощения выполним подстановку конкретных значений для произвольных постоянных

```
> subs(C=-k,k=1,_C1=1,_C2=1,sol);
```

$$\left[-\frac{1}{2}I\left(-1+e^{(-2I(x+1))}\right)e^{(Ix+I-t)},\;-\frac{1}{2}I\left(-1+e^{(2I(x+1))}\right)e^{(-t-Ix-I)}\right]$$

Преобразуем к тригонометрическому виду

```
> convert(",trig);
```

$$\left[-\frac{1}{2}I\left(-1+\cos(2x+2)-I\sin(2x+2)\right)(\cosh(t)-\sinh(t))(\cos(x+1)+\right.$$
$$+I\sin(x+1)),$$
$$-\frac{1}{2}I\left(-1+\cos(2x+2)-I\sin(2x+2)\right)(\cosh(t)-\sinh(t))(\cos(x+1)-$$
$$\left.-I\sin(x+1))\right]$$

и упростим

```
> S:=combine(");
```

$$S:=[-\cosh(t)\sin(x+1)+\sinh(t)\sin(x+1),\;\cosh(t)\sin(x+1)-$$
$$-\sinh(t)\sin(x+1)\;]$$

Теперь можно построить график первого решения (рис. 9)

```
> plot3d(S[1],x=-5..5,t=0..5);
```

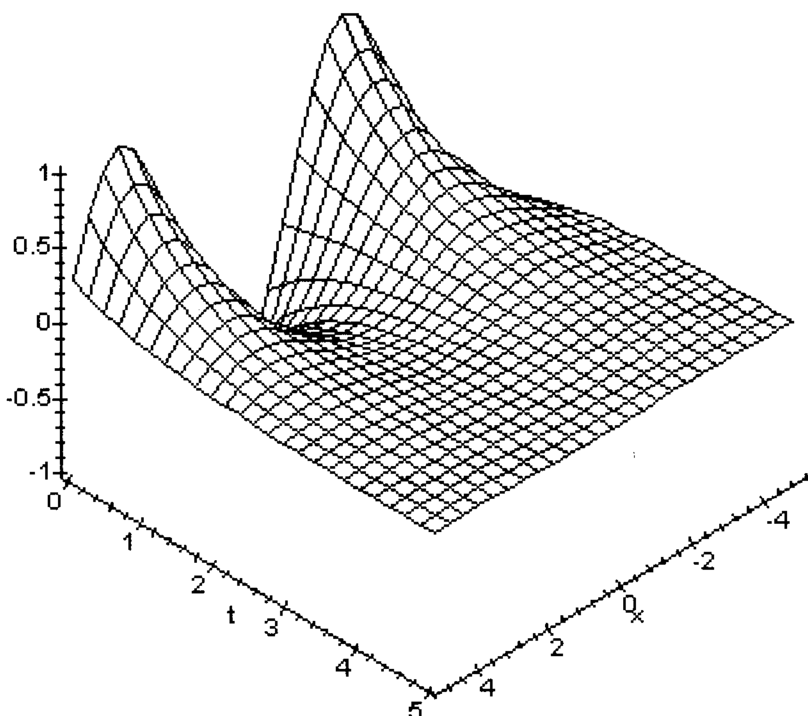

Рис. 9

Проверим правильность первого решения

```
> simplify(subs(u(x,t)=sol[1],heat));
```

$$0 = 0$$

В качестве еще одного примера рассмотрим волновое уравнение

```
> restart;wave:=diff(u(x,t),t,t)-c^2*diff(u(x,t),
x,x);
```

$$wave := \left(\frac{\partial}{\partial t} \mathrm{u}(x, t)\right) - c^2 \left(\frac{\partial^2}{\partial x^2} \mathrm{u}(x, t)\right)$$

Найдем решение для u(x,t).

```
> sol:=pdesolve(wave,u(x,t));
```

$$sol := \mathrm{u}(x, t) = _\mathrm{F1}(t\, c + x) + _\mathrm{F2}(t\, c - x)$$

Здесь _F1 и _F2 — произвольные функции. Заменим их конкретными функциями f1 и f2

```
> f1:=xi -> sech(-xi^2);
```

$$f1 := \xi \to \operatorname{sech}(-\xi^2)$$

```
> f2:=xi -> piecewise(-1/2<xi and xi<1/2,1,0);
```

$$f2 := \xi \to \text{piecewise}\left(\frac{-1}{2} < \xi \ \textbf{and}\ \xi < \frac{1}{2}, 1, 0\right)$$

Заменим наименования функций в решении на f1 и f2, а также положим c=1.

```
> subs(_F1=f1, _F2=f2, c=1, sol);
```

$$u(x, t) = f1(t + x) + f2(t - x)$$

Теперь подставим значения f1 и f2 в u(x,t), чтобы получить конкретное решение.

$$\operatorname{sech}((t + x)^2) + \begin{pmatrix} 1 & -\frac{1}{2} - t + x < 0 \ \textbf{and}\ t - x - \frac{1}{2} < 0 \\ 0 & \textit{otherwise} \end{pmatrix}$$

```
> subs(",u(x,t))";
```

Применим команду **unapply**, чтобы превратить полученное выражение в функцию от x и t

```
> f:=unapply(",x,t);
```

$$f := (x, t) \to \operatorname{sech}((t + x)^2) + \text{piecewise}\left(-\frac{1}{2} - t + x < 0 \ \textbf{and}\ t - x - \frac{1}{2} < 0, 1, 0\right)$$

Теперь мы можем построить график решения (рис. 10).

```
> plot3d(f, -8..8, 0..5, grid=[60,60]);
```

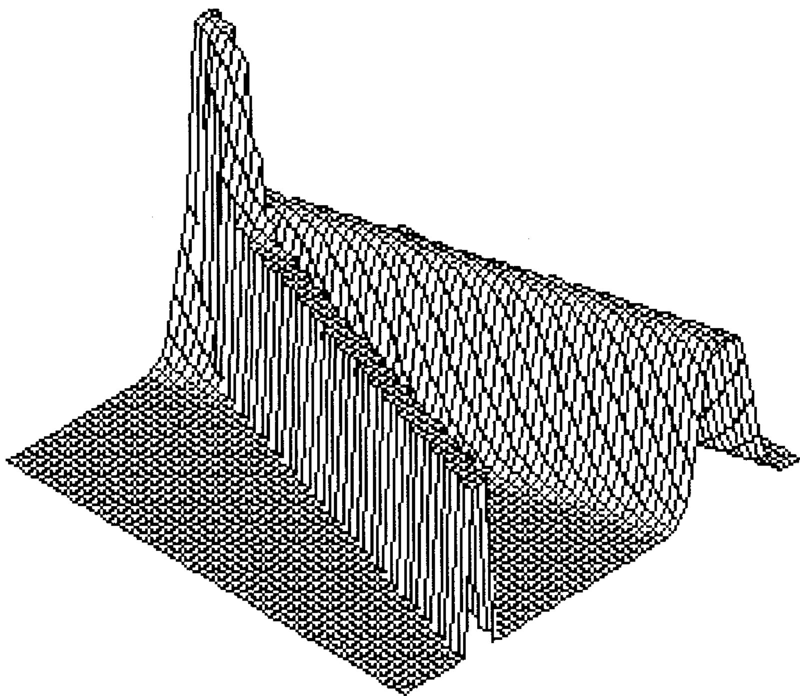

Рис. 10

На графике мы видим две волны, которые представляют решение волнового уравнения.

7. Графики и анимация в *Maple*

Пожалуй, одно из наиболее впечатляющих свойств программы *Maple* — превосходная графика. Команды построения графиков и анимации *Maple* позволяют удовлетворить большинство научных и инженерных потребностей, могут служить прекрасной иллюстрацией в учебном процессе.

Программа имеет большое количество функций и опций настроек для построения как двух-, так и трехмерных графических объектов. Помимо команд **plot** и **plot3d** основной библиотеки имеется несколько специализированных пакетов для этих целей:

- это прежде всего пакет **plots**, содержащий около пятидесяти команд для построения различного рода графиков и анимации;
- вспомогательный пакет **plottools**, позволяющий создавать различные (около тридцати) двух- и трехмерные графические примитивы, которые могут быть применены в других графиках;
- пакет **stats[statplot]**, содержащий команды для построения специализированных статистических графиков; пакет **DEtools**, содержащий команды построения графиков решения дифференциальных уравнений, как обыкновенных так и в частных производных, фазовых портретов, полей направлений;
- и, наконец, геометрический пакет **geometry**, содержащий команду **draw**, позволяющую отобразить различные геометрические построения на плоскости.

Версия 4 программы *Maple* поддерживает 45 систем координат (в предыдущей версии всего 4), появились также команды **changecoords** и **addcoords**, позволяющие пользователю переходить от одной системы координат к другой, а также вводить свои системы координат.

Многие функции настройки осуществляются непосредственно с инструментальной панели программы (задание стиля, цвета, подсветки, перспективы, вида координатных осей), но могут вводиться непосредственно в команду. На следующих примерах будет проиллюстрировано сказанное.

7.1. Двухмерные графики

Графики на плоскости можно строить при помощи команды **plot** либо командами уже упомянутых других пакетов.

В двухмерные графики можно включать дополнительные опции:

- опция **numpoints** позволяет изменять количество точек графика. Значение этой опции по умолчанию — 49;
- опцией **color** можно задать цвет точек графика;
- опцией **title** — добавить заголовок (см. рис. 12.);
- опцией **axes** задается тип осей (рамка (**FRAME**), прямоугольник (**BOXED**), ортогональные (**NORMAL**) или без осей (**NONE**));

- опции **xtickmarks** и **ytickmarks** управляют числами меток на осях;
- опция **style** применяется для задания интерполяции кривой по заданным точкам (**line** — выводится интерполяционная кривая, **point** — выводятся точки).

Графики, построенные при помощи команды *plot*

График явно заданной функции (рис. 11)

```
> plot(x*sin(x),x=-3*Pi..3*Pi);
```

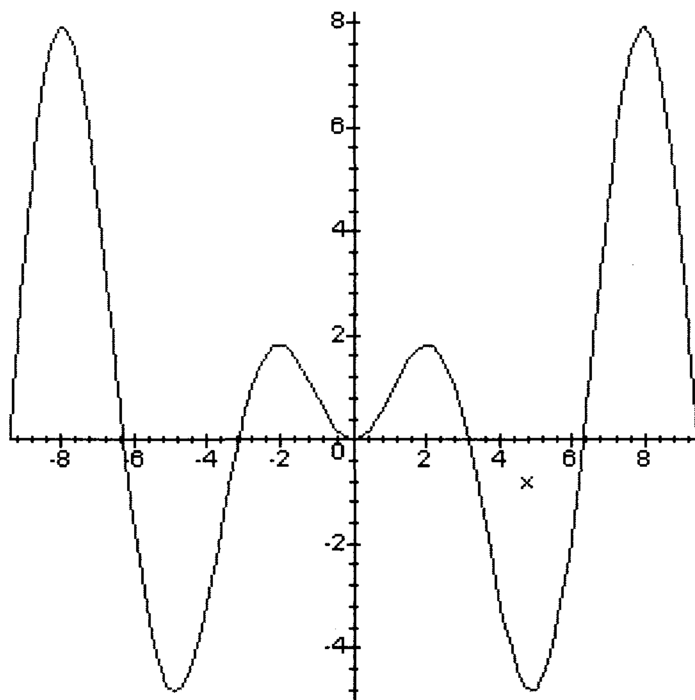

Рис. 11

График функции, заданной в параметрической форме (рис. 12)

```
> plot([sin(2*t),cos(3*t),t=0..2*Pi], color=BLUE,
  title= `МОЙ СИНИЙ ГРАФИК`);
```

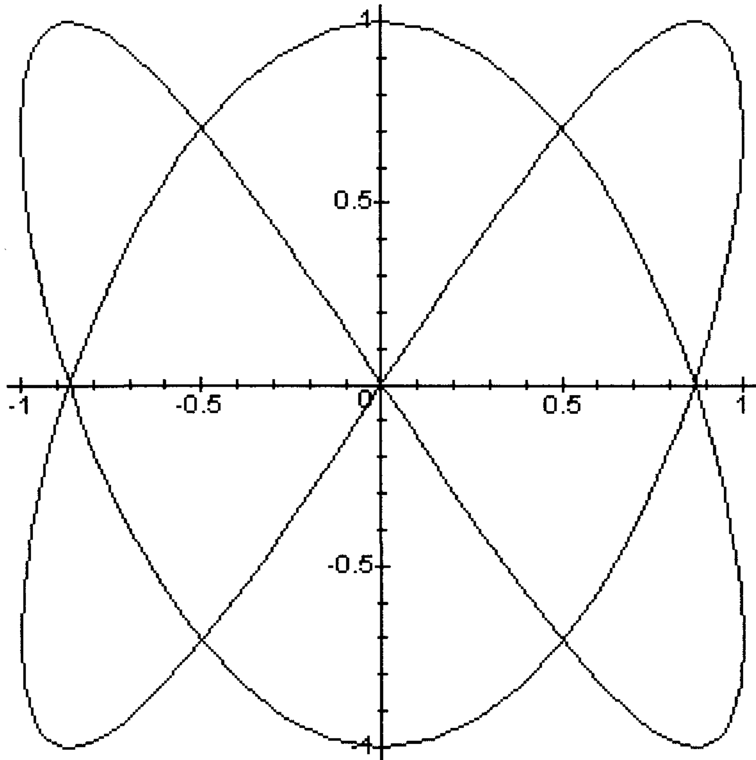

Рис. 12

Графики функций, заданных в виде процедур или операторов (рис. 13)

```
> F:=proc(x) sin(exp(x))+sqrt(abs(x)) end;
```

F:=**proc**(x) sin(exp(x))+sqrt(abs(x)) **end**

```
> plot(F, -Pi..Pi);
```

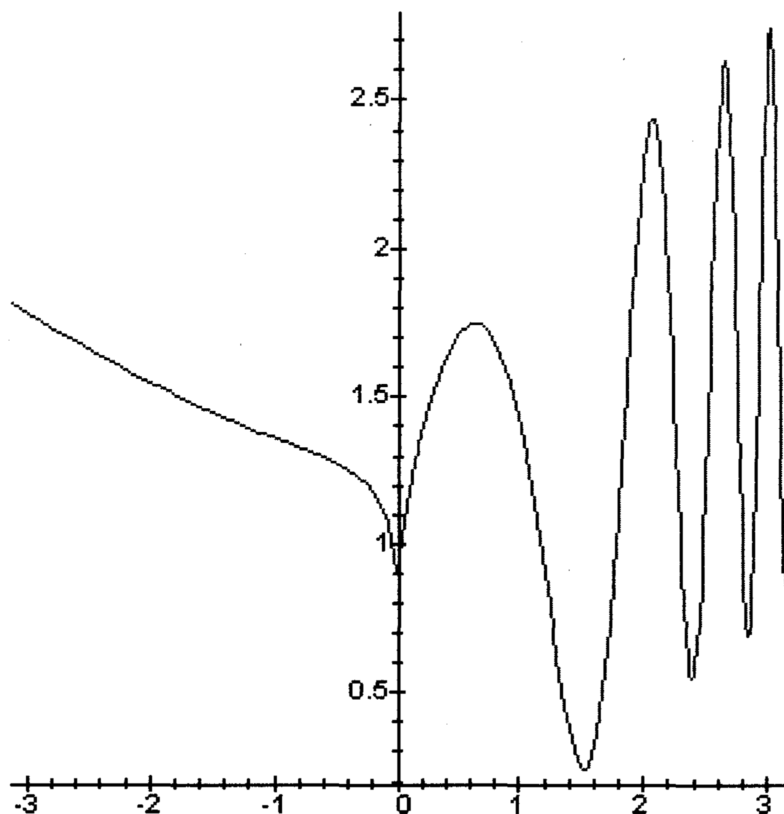

Рис. 13

Для выражений, имеющих бесконечные разрывы, можно добавить опцию **discont=true** (рис. 14).

```
> plot(ln(1+tan(x)),x=-2*Pi..2*Pi,discont=true);
```

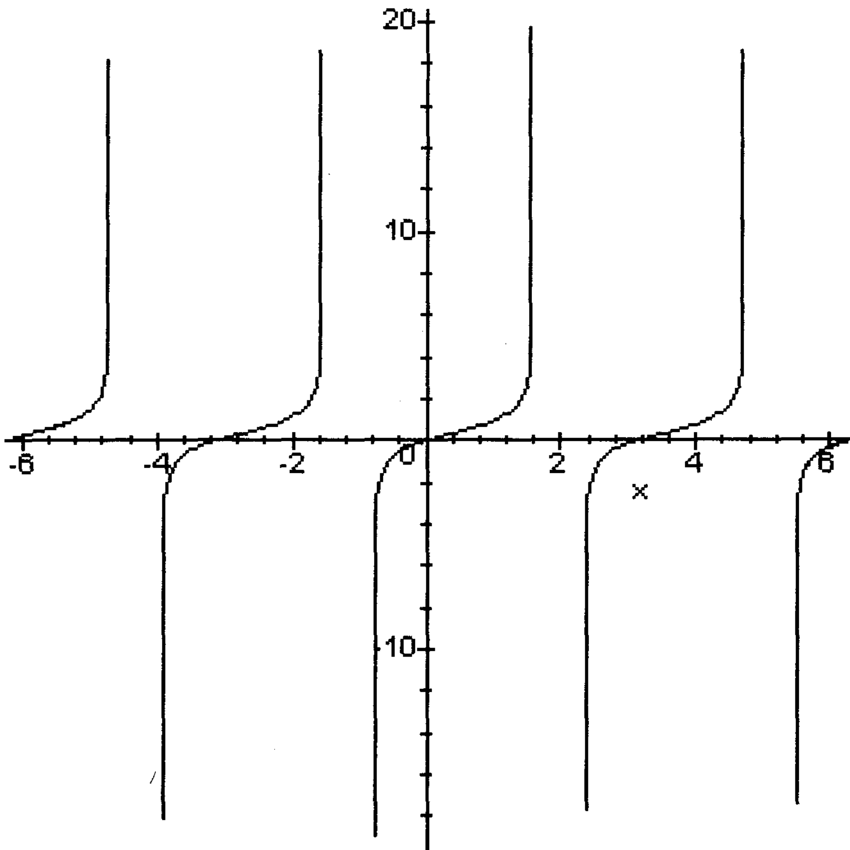

Рис. 14

Несколько графиков объединяются в набор или список (рис. 15)

```
> plot([sin(x), convert(series(sin(x),x), polynom)],
  x=0..Pi, color=[red,blue], style=[line,point]);
```

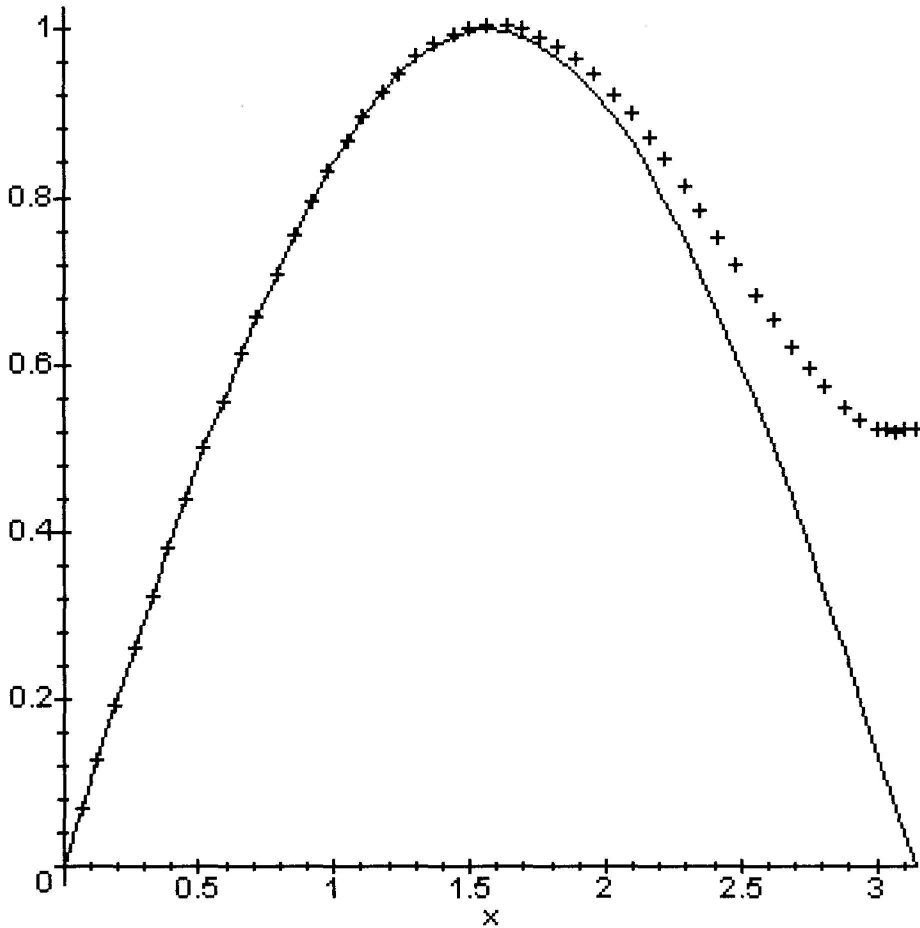

Рис. 15

Бесконечно протяженный график (рис. 16)

```
> plot(ln(1+sin(x)), x=0..infinity,-3..3);
```

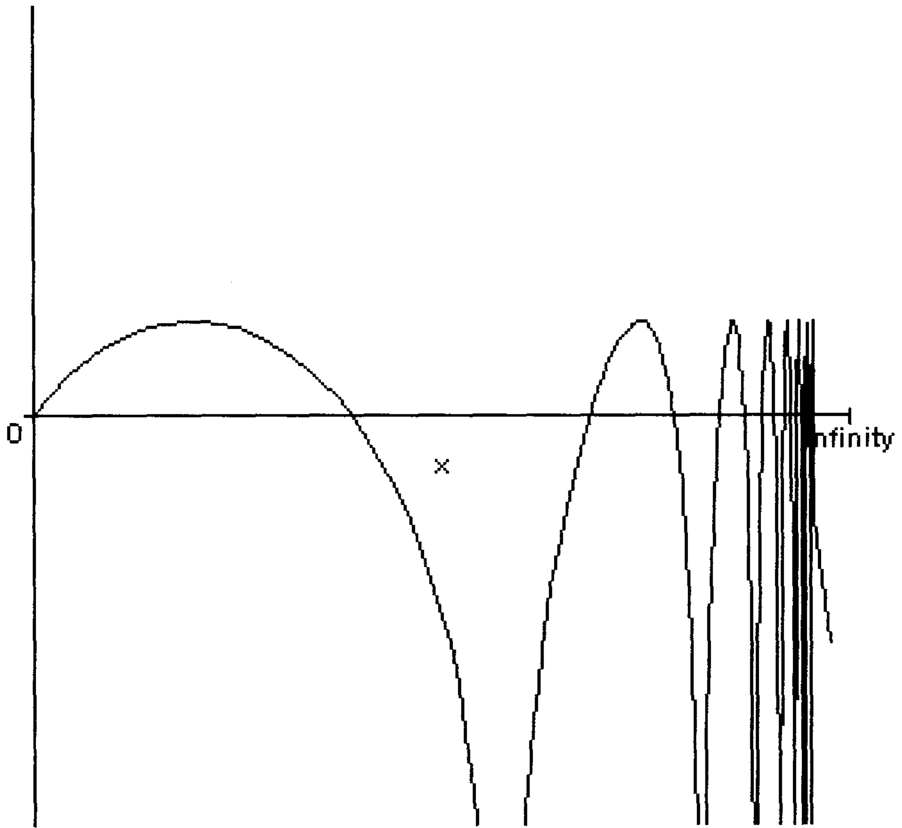

Рис. 16

График, построенный по заданным точкам (рис.17)

```
> l := [[ n, FresnelC(n)] $n=1..30]:
> plot(l, x=0..15, style=point, symbol=cross);
```

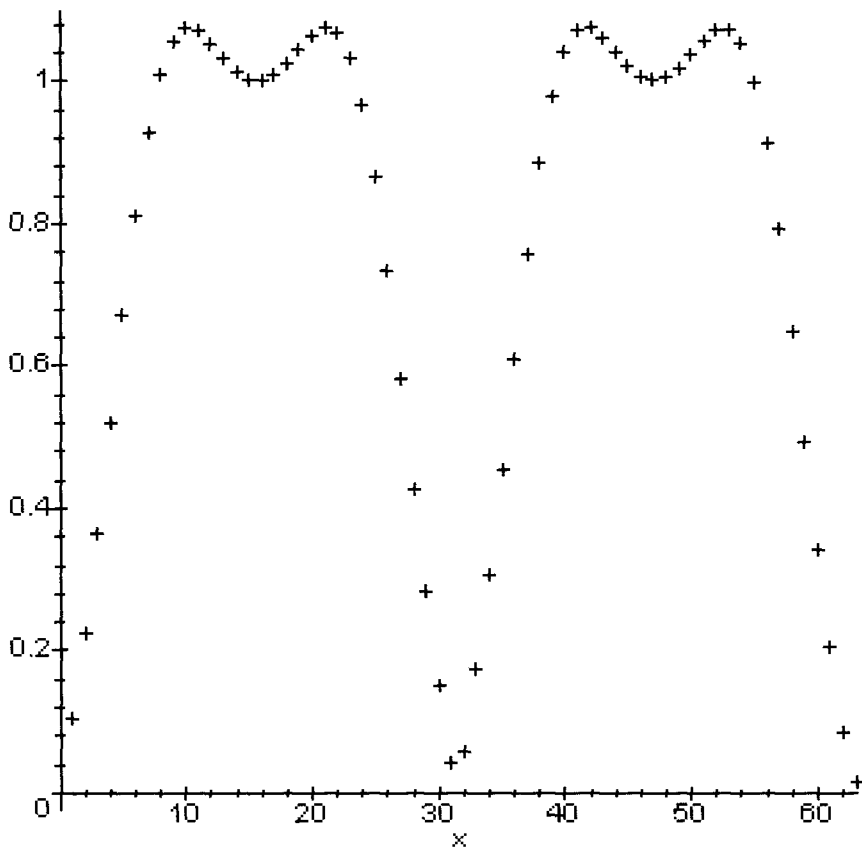

Рис. 17

Полярные координаты с заданной толщиной линии (рис. 18)

```
> plot([sin(3*x),x,x=0..2*Pi],coords=polar,
  thickness=2);
```

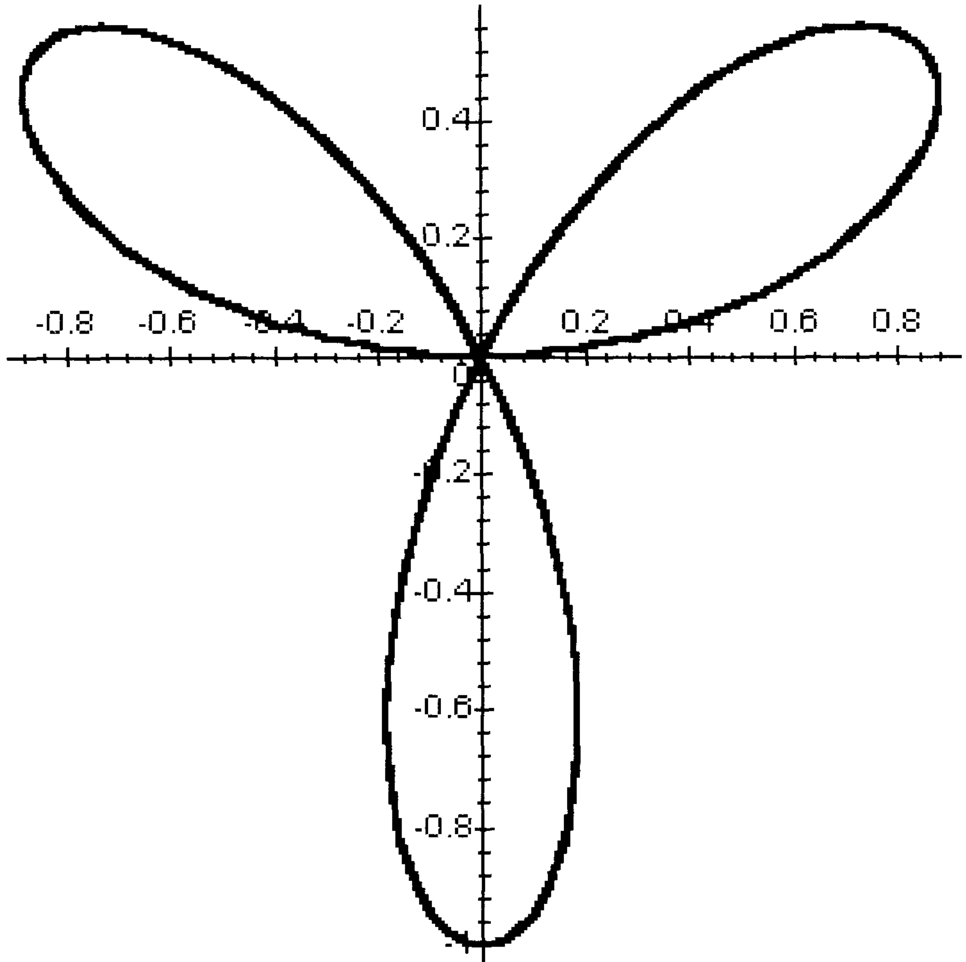

Рис. 18

Графики, построенные при помощи команд пакета *plots*

Команды пакета **plots** расширяют количество типов двухмерных графиков. Так строится график конформного отображения (рис. 19)

```
> with(plots):
  conformal((z-1)^(1/2)*(z+1)^(1/2),z=-1-I..1+I);
```

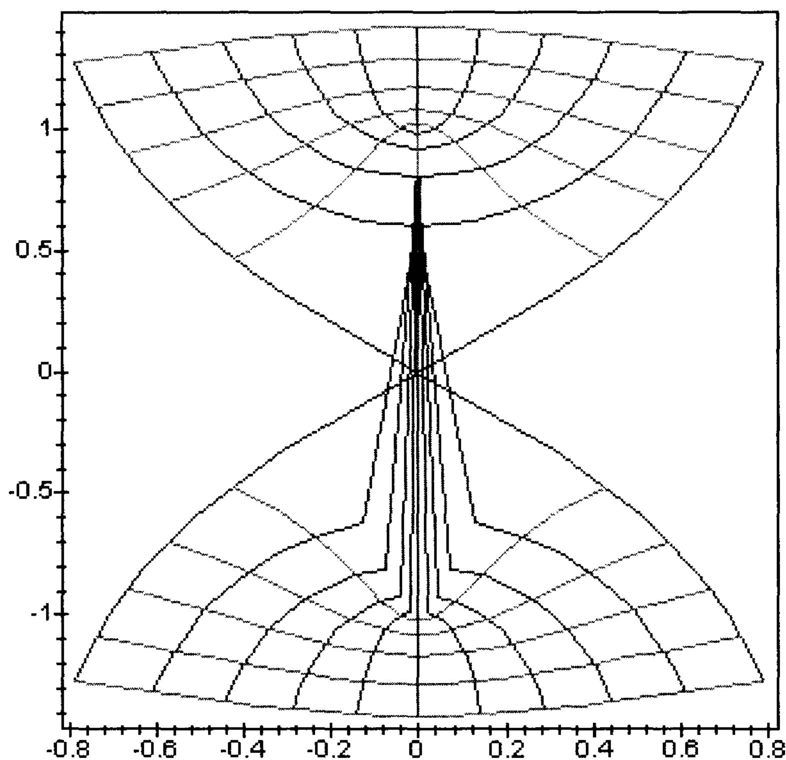

Рис. 19

Контурный график, отображающий линии пересечения поверхности с плоскостями, задаваемыми опцией **contours** (рис. 20)

```
> contourplot(sin(x*y),x=-Pi..Pi,y=-Pi..Pi,
  grid=[15,15],contours=[-0.9,-1/2,0,1/2,0.9]);
```

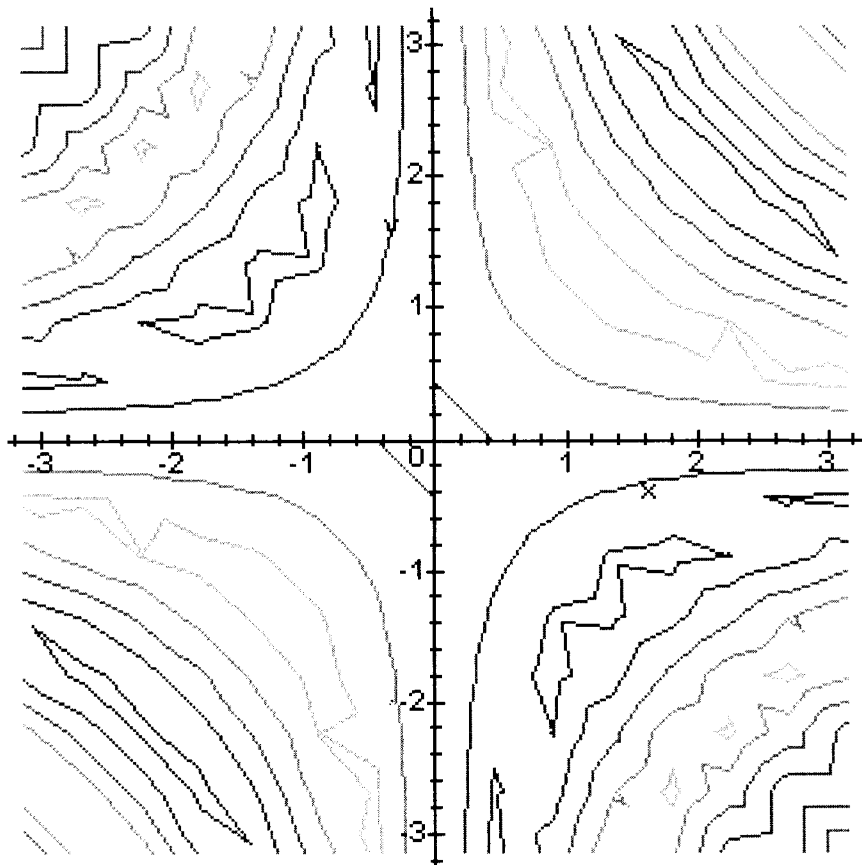

Рис. 20

На следующем рисунке для той же функции представлен график плотности линий уровня (более темные участки соответствуют большей плотности (рис. 21)

```
> densityplot(sin(x*y),x=-Pi..Pi,y=-Pi..Pi,
  axes=boxed);
```

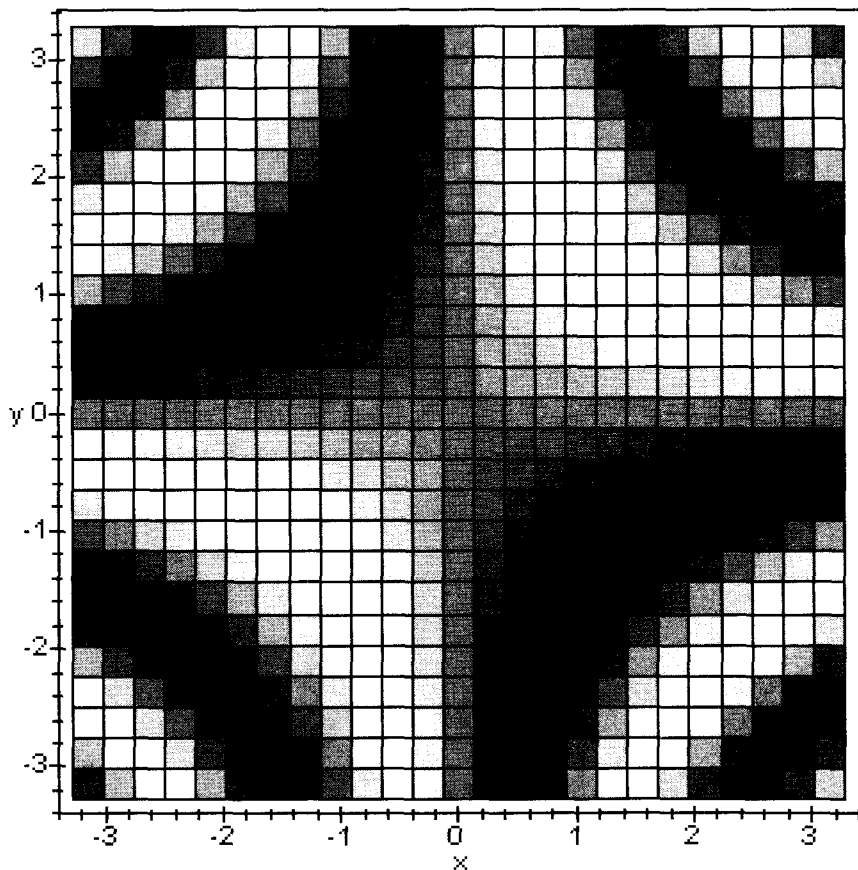

Рис. 21

График векторного поля градиентов той же функции (рис. 22)

```
> gradplot(sin(x*y),x=-Pi..Pi,y=-Pi..Pi,
  arrows=SLIM);
```

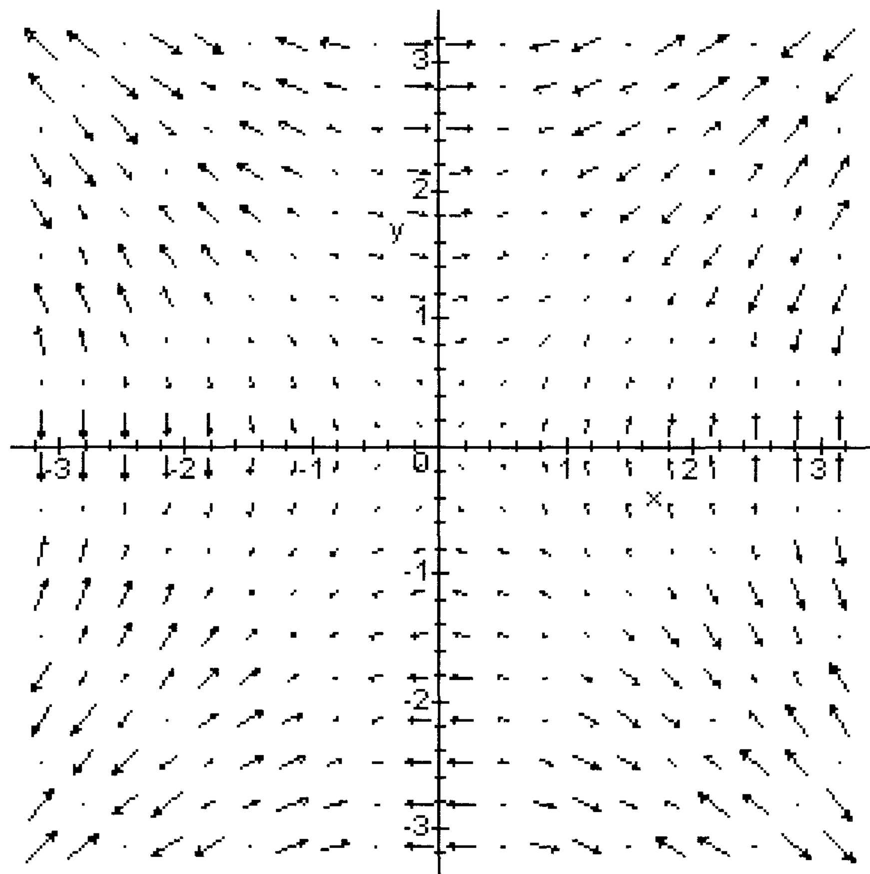

Рис. 22

График двухмерного векторного поля (рис. 23)

```
> plots[fieldplot] ( [cos(x)*sin(y), cos(y)*sin(x)],
  x=-Pi ..Pi, y=-Pi ..Pi, arrows=SLIM);
```

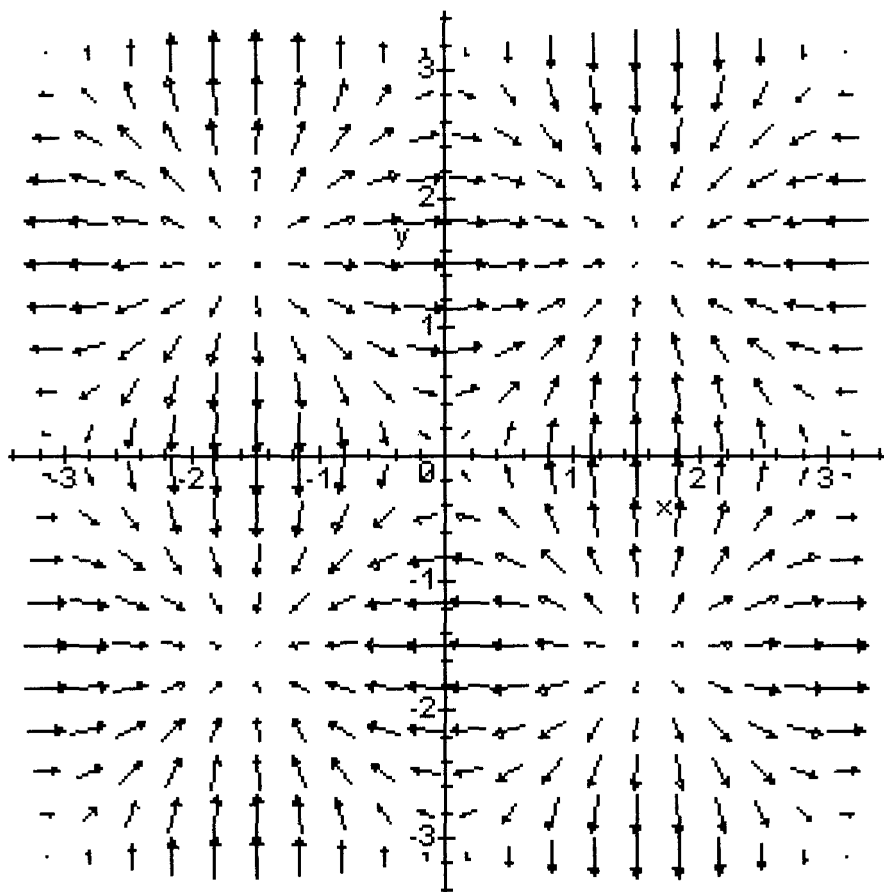

Рис. 23

На следующем рисунке представлен график неявно заданной функции (рис. 24)

```
> plots[implicitplot]((x^2/25)+(y^2/9)=1, x=-6 ..6,
  y=-6 ..6,scaling=CONSTRAINED);
```

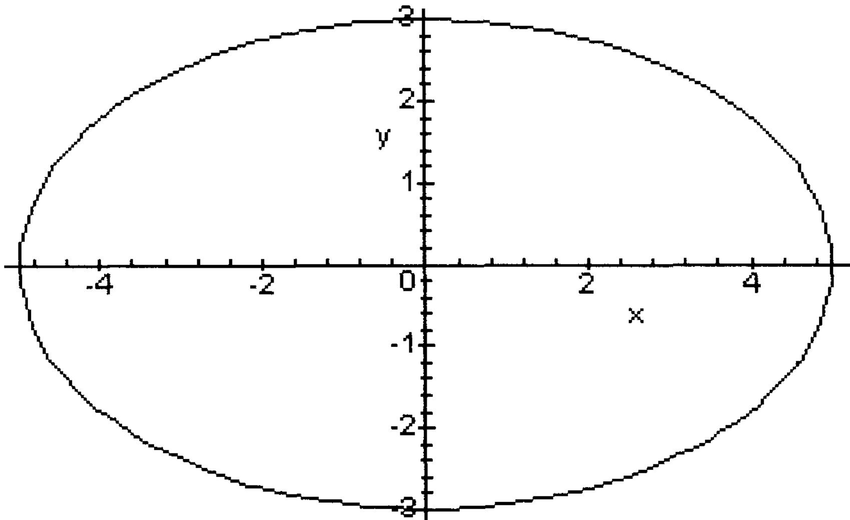

Рис. 24

График области, удовлетворяющей неравенствам; задаются цвета открытых и закрытых границ, внешней и внутренней областей, а также толщина линий границ (рис. 25)

```
> plots[inequal]( {a+b>3, 2*b-a<6, 3*a+2*b>5,
  -b+a<=8, 3*a+2*b>0},
  a=-10..30, b=-10..15, optionsfeasi ble=(color=red),
     optionsopen=(color=blue, thickness=2),
     optionsclosed=(color=green, thickness=3),
     optionsexcluded=(color=yellow) );
```

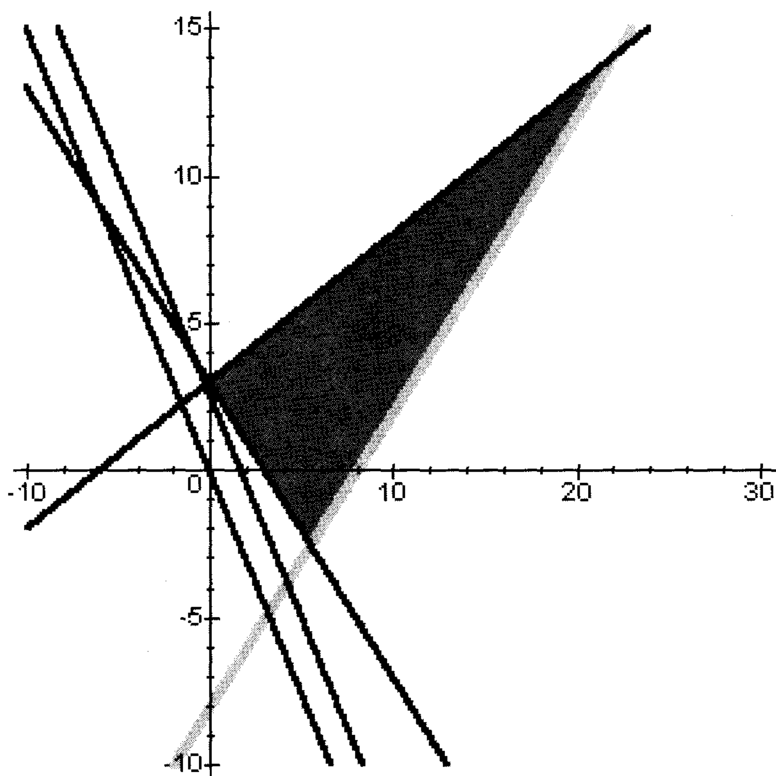

Рис. 25

Следующей командой строится график списка точек, прочитанный из первого столбца файла *Data3.txt* (рис. 26)

```
> plots[listplot](readdata('e:\\MapleV4\\data3.txt',
  float,1), color=gold);
```

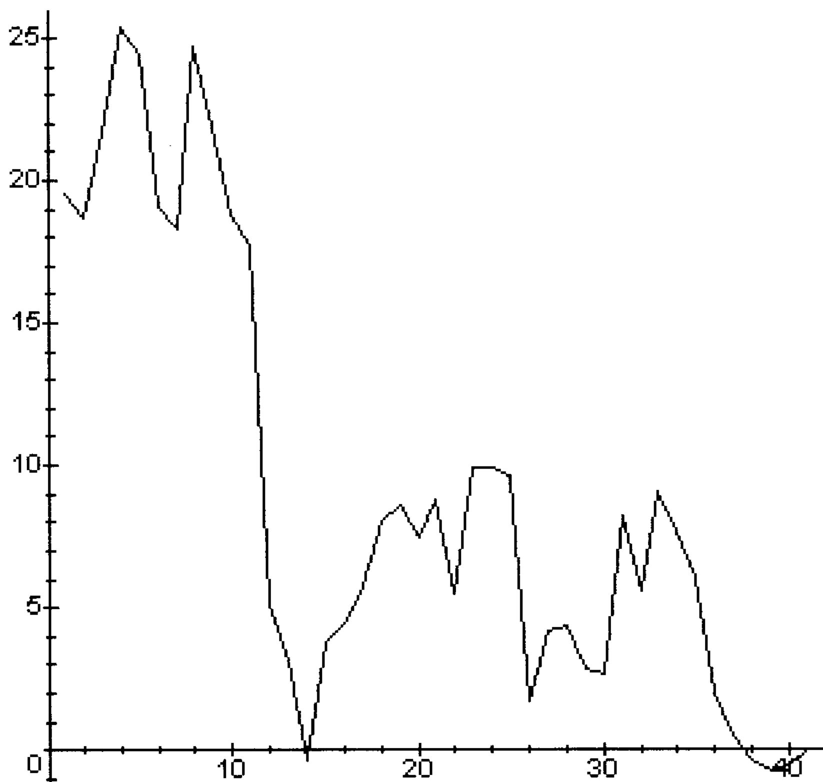

Рис. 26

Возможно построение графиков в логарифмической и двойной логарифмической шкалах, например (рис. 27)

```
> plots[loglogplot]({x->exp(sin(x)), x->exp(cos(x))},
  1..10);
```

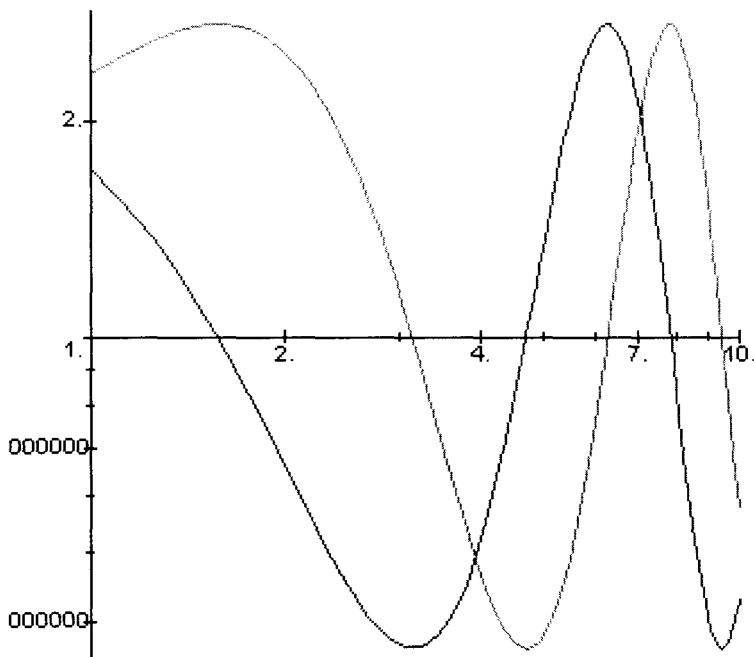

Рис. 27

В пакете имеется команда **odeplot** для построения графика решения дифференциального уравнения.

```
> f1:=diff(y(x),x,x,x)+x*sqrt(abs(diff(y(x),x)))+
x^2*y(x);
```

$$f1 := \frac{\partial}{\partial t}\mathrm{y}(x) + x\sqrt{\left|\frac{4\partial^2}{\partial x^2}\mathrm{y}(x)\right|} + x^2\, y(x)$$

```
> F1:=dsolve({f1,y(0)=0,D(y)(0)=1,D(D(y))(0)=1},
  y(x),numeric);
```

$F1$:= **proc**($rkf45_x$) ... **end**

```
> p:=odeplot(F1,[x,y(x)],-4..5):
```

К графику можно добавить надписи при помощи команды **textplot**

```
> t1 := textplot([2,3+delta,'Local Maxima (2, 3)'],
  align=ABOVE):
  t2 := textplot([3.9,-14-delta,'Local Minima (3.9,
  -14)'], align=BELOW):
```

Теперь при помощи команды **display** отобразим все построенные графические объекты на одном графике (рис. 28)

```
> plots[display]({p,t1,t2});
```

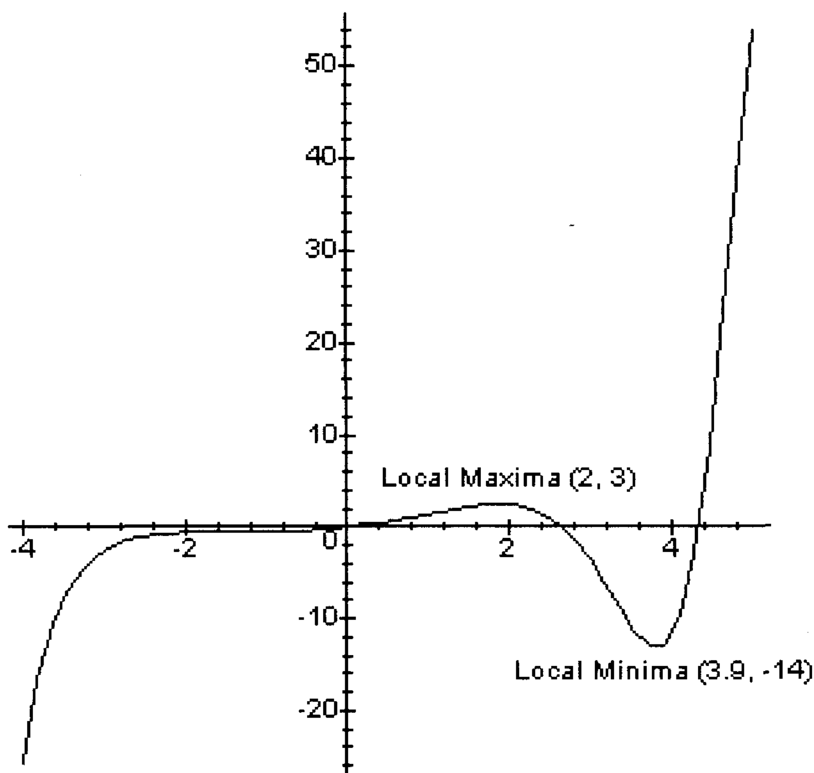

Рис. 28

Мультфильмы на плоскости строятся при помощи команды **animate** пакета **plots** (рис. 29)

```
> plots[animate]([sin(t*(2+u)),coc(t*(3+u)),
  t=o..2*pi], u=0..10,colour=red);
```

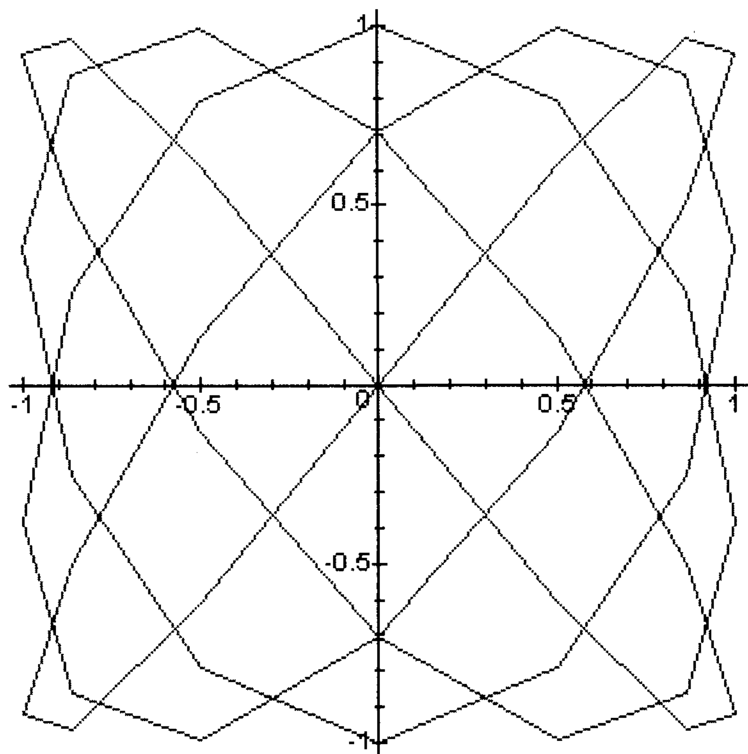

Рис. 29

Пакет содержит также команду **coordplots**, позволяющую строить различные системы координат, что позволяет на одном графике представить вид системы координат и сам графический объект, построенный в этой системе координат. На плоскости имеются следующие виды систем координат: биполярная (**bipolar**), кардиоидная (**cardiod**), прямоугольная (**cartesian**), Кассини (**cassinian**), эллиптическая (**elliptic**), гиперболическая (**hyperbolic**), инверсная Кассини (**invcassinian**), инверсная эллиптическая (**invelliptic**), логарифмическая (**logarithmic**), Максвелла (**maxwell**), параболическая (**parabolic**), полярная (**polar**), роза (**rose**) и тангенциальная (**tangent**).

Приведем примеры (рис. 30, рис. 31)

```
> a:=plot(sin(x)^2-cos(x)^2, x=0..2*Pi, coords=polar,
  thickness=0):
> b := coordplot(polar,[0..1.5,0..2*Pi]):
> display([a,b]);
```

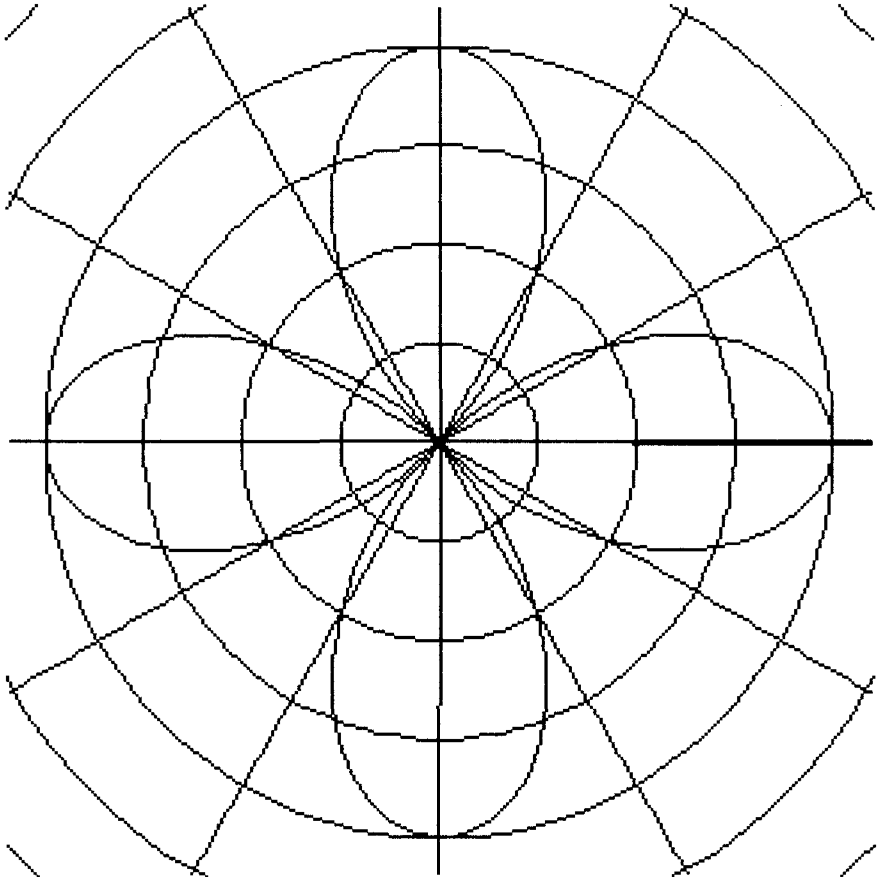

Рис. 30

```
> r1 := plot(sin(cos(x)),x=-2*Pi..2*Pi,
  coords=bipolar,  thickness=3):
  r2 := coordplot(bipolar):
  plots[display]([r1,r2]);
```

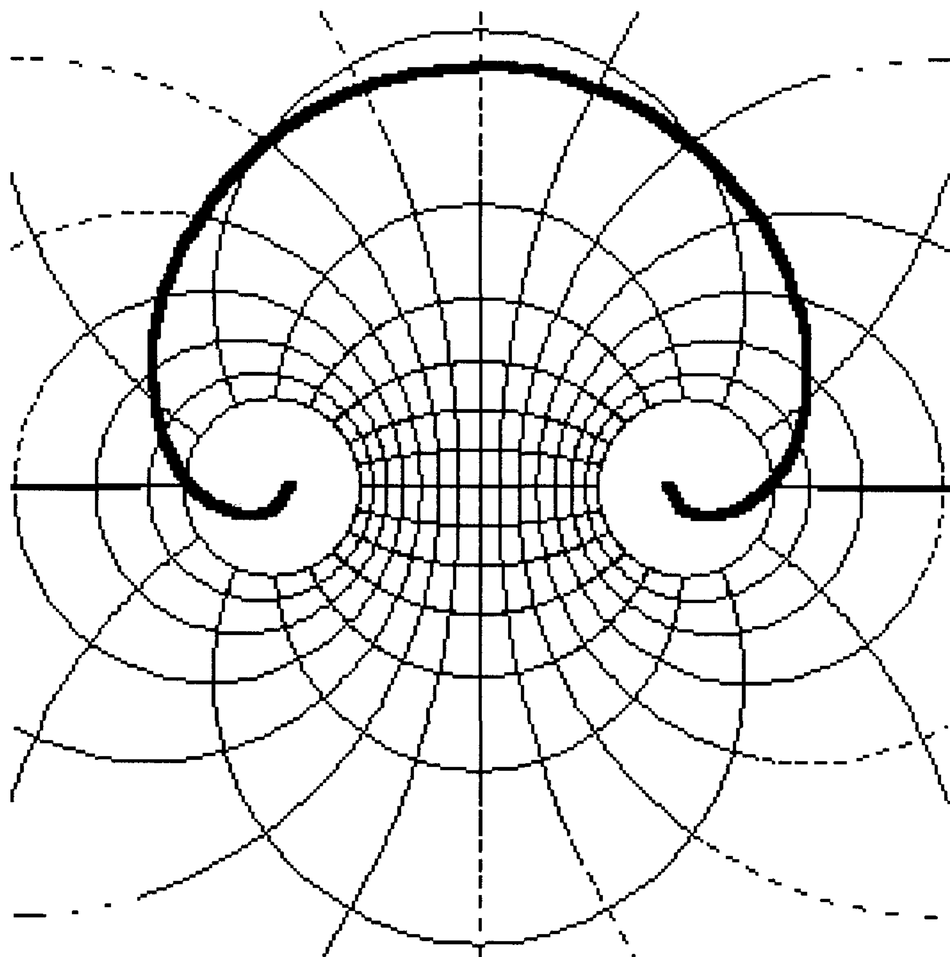

Рис. 31

Графика пакета *plottools*

Как уже упоминалось выше, команды этого пакета позволяют строить различные графические примитивы, которые в дальнейшем могут быть использованы в других графиках, а также производить различные перемещения фигур. На приведенном примере построены окружность и многоугольник и при помощи команды **rotate** получено несколько расположенных по окружности фигур (рис. 32)

```
> with(plottools):
> c := circle([1,1], 0.5, color=red):
> l := polygon([[0,0], [3,4], [3,1], [2,2],
  [0,5]], color=yellow, linestyle=3, thickness=2):
> r1:=seq(rotate(c, Pi*i/3), i=1..6):
> r2:=seq(rotate(l, Pi*2*i/3), i=1..3):
> display(r1, r2);
```

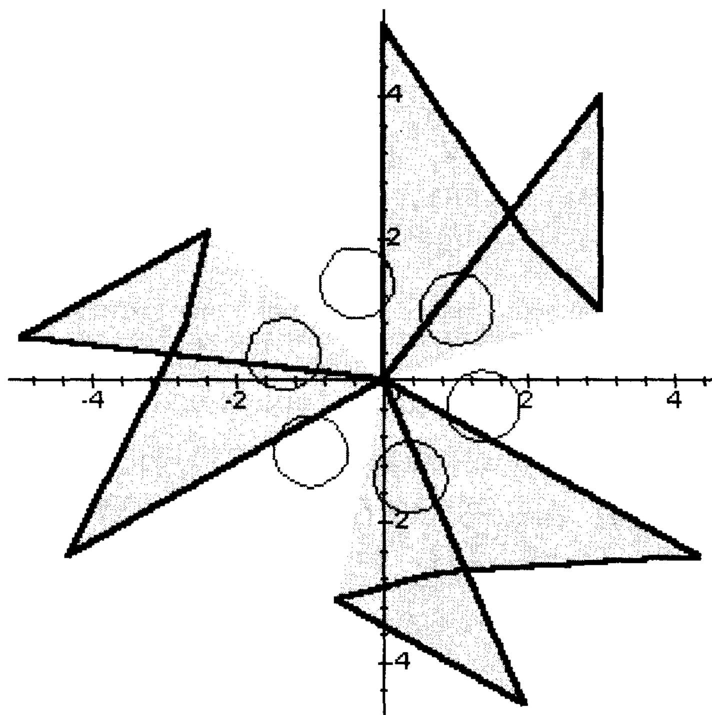

Рис. 32

Графика статистического пакета

Пакет **stats[statplots]** содержит следующие команды, позволяющие строить различные статистические графики на плоскости:

```
boxplot   histogram notchedbox
quantile  quantile2 scatter1d
scatter2d symmetry
```

Пусть, например, имеются две серии статистических данных, независимая Xdata и зависимая Ydata.

```
> with(stats):
with(stats[statplots]):
> Xdata:= [4.535, 4.029, 5.407, 1.605, 5.757,
          3.527, 7.890, 8.159, 6.092, 13.442,
          2.845, 5.172, 3.277, 8.810, 3.657,
          7.226, 3.851, 2.162, 2.668, 4.692]:

> Ydata:= [7.454, 4.476, 2.873, 5.476, 9.975, -
          1.476, 1.033, 1.140, 4.813, .450,  -
          .788, 9.389, 4.811, -3.107, 4.407,
          5.534, 1.691, -.789, 1.684, 1.605]:
Warning, new definition for transform
```

Построим статистический график рассеяния с прямоугольными диаграммами (рис. 33)

```
> plots[display]({statplots[scatter2d]
  (Xdata, Ydata),
  statplots[boxplot[15]](Ydata), statplots
  [xyexchange] (statplots[notchedbox[12]](Xdata))},
  view =[0..17,-4..14], axes=FRAME);
```

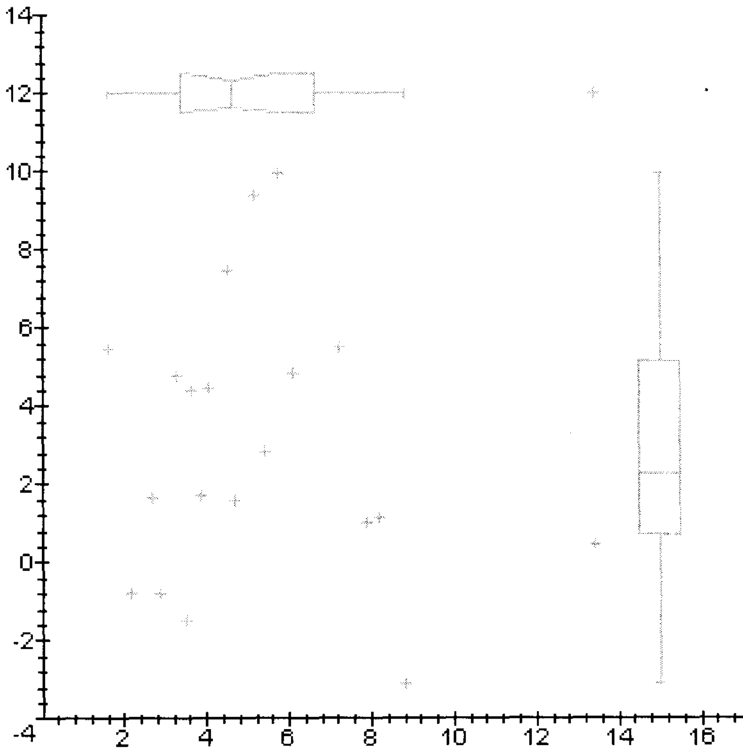

Рис. 33

На следующем графике построены гистограмма по данным Xdata и кривая нормального распределения со средним μ и дисперсией σ полученным из Xdata (рис. 34).

```
> mu:=sum(Xdata[i],i=1..nops(Xdata))/(nops(Xdata)-1);
  sigma:=sqrt(sum((Xdata[i]-mu)^2,
  i=1..nops(Xdata))/ (nops(Xdata)-1));
```

$$\mu := 5.515947369$$
$$\sigma := 2.815158873$$

```
> histogram(Xdata, colour=yellow):
  plot(stats[statevalf,pdf,normald[mu,sigma]],
  -3..15, color=red):
  plots[display]({",""});
```

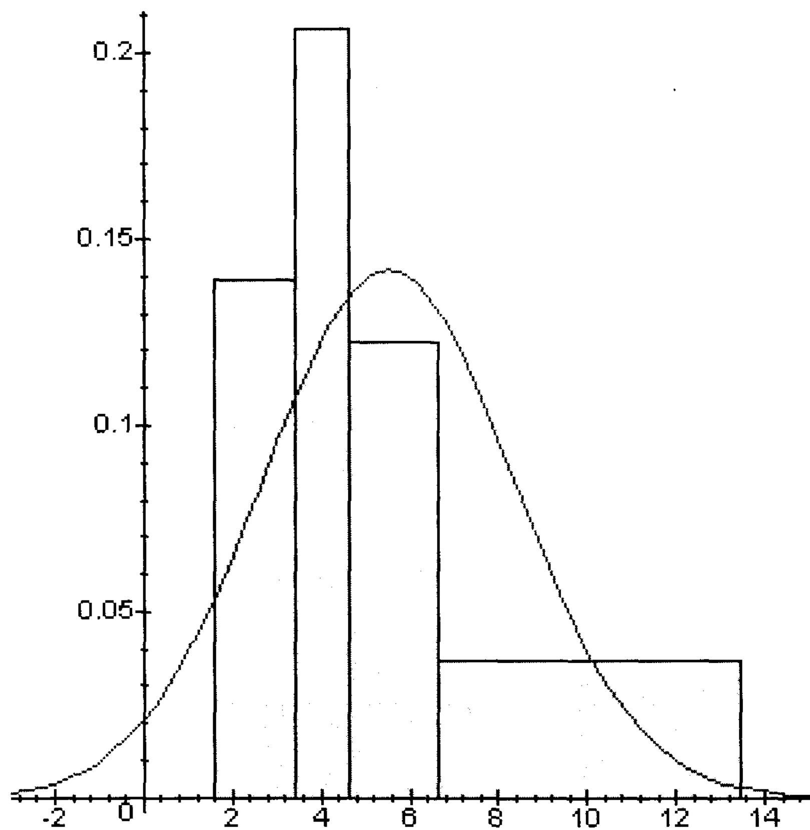

Рис. 34

Графика пакета *DEtools*

Команда **DEtools[DEplot] (deqns, vars, trange, inits, xrange, yrange, eqns)** — строит решение обыкновенных дифференциальных уравнений и систем, она аналогична команде **odeplot** пакета **plots**, но гораздо более функциональна.

Параметры аргумента имеют следующее значение:

- deqns — список или набор обыкновенных дифференциальных уравнений любого порядка;
- vars — список зависимых переменных;
- trange — диапазон независимой переменной;
- inits — начальные условия (если они не указаны, то строится только поле направлений);
- yrange — диапазон первой зависимой переменной;
- xrange — диапазон второй зависимой переменной;
- eqns — равенства ключевое слово=величина, задающие дополнительные опции.

По заданному набору или списку начальных условий и системы дифференциальных уравнений первого порядка или одного дифференциального уравнения более высокого порядка **DEplot** строит кривые решения численными методами. Для двух переменных решения системы первого порядка будет также строиться график поля направлений, при условии, что система автономна. Для неавтономных систем поле направлений не будет строиться (в этом случае возможны только кривые решения). В любом случае должна быть только одна независимая переменная.

Метод интегрирования по умолчанию — классический метод *Рунге-Кутта*. Другие методы должны быть указаны явно в опциях команды. Заметим, что поскольку для создания кривых используются численные методы, вид графика может зависеть от метода интегрирования, особенно когда имеются асимптоты.

Представляемое поле направлений состоит из сетки стрелок, касательных к кривым решения. Для каждой точки сетки стрелка с центром в (x,y) будет иметь наклон dy/dx. Этот наклон вычисляется по формуле (dy/dt)/(dx/dt), причем обе производные заданы первым аргументом **DEplot**. Система автономна, когда все члены и множители, кроме производных, не содержат в явном виде аргументов, содержащих независимую переменную.

Для одного дифференциального уравнения более высокого порядка могут быть построены только кривые решения.

По умолчанию, две зависимые переменные будут построены, если не указано иное в опции **scene**.

Ключевые слова опций могут быть следующими:

- 'arrows' = тип стрелки ('SMALL', 'MEDIUM', 'LARGE', 'LINE', or 'NONE');
- 'colour' = цвет стрелки, который может быть задан различными способами;
- 'dirgrid' = массив, устанавливающий число точек сетки, по умолчанию [20,20];

- 'iterations' = число итераций (натуральное число);
- 'linecolour' = цвет линии, задаваемый различными способами;
- 'obsrange' = TRUE, FALSE, устанавливает, прерывать ли вычисление, если кривая выходит из обзора;
- 'scene' = [имя, имя], определяет какие зависимые переменные и в каком порядке должны быть выведены в график;
- 'stepsize' = определяет расстояние между точками, которое используется при вычислении точек графика, для trange=a..b, по умолчанию h = abs((b−a))/20.

Приведем примеры. Следующий график (рис. 35) в точности повторяет график, построенный при помощи команды **odeplot** пакета **plots** (смотрите выше)

```
> with(DEtools):
  DEplot(diff(y(x),x,x,x)+x*sqrt(abs(diff(y(x),
  x)))+ x^2*y(x),
  {y(x)},x=-4..5,[[y(0)=0,D(y)(0)=1,
  (D@@2)(y)(0)=1]], stepsize=.1, linecolour=red);
  Warning, new definition for transform
```

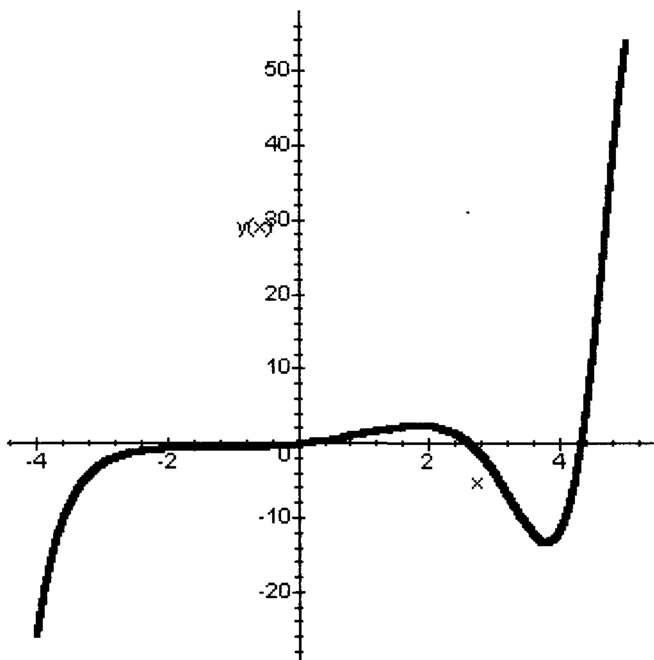

Рис. 35

Следующий пример системы из трех уравнений первого порядка — строится фазовая кривая для переменных z и y, цвет кривой задан функцией от независимой переменной, задан также метод решения системы (рис. 36).

```
> DEplot({D(x)(t)=y(t)-z(t), D(y)(t)=z(t)-x(t),
  D(z)(t)=x(t)-y(t)*2},
  {x(t),y(t),z(t)},t=-2..2, [[x(0)=1, y(0)=0,z(0)=2]],
  stepsize=.05,scene=[z(t), x(t)],
  linecolour=sin(t*Pi/2), method=classical
  [foreuler]);
```

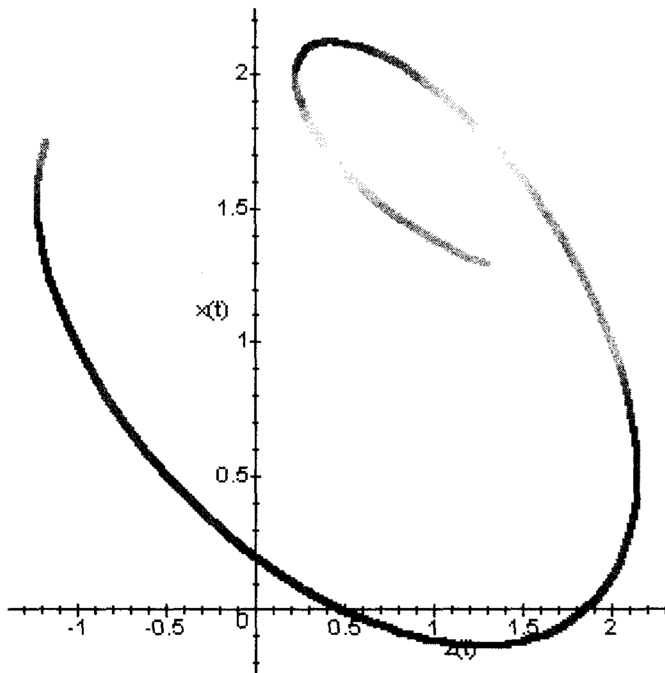

Рис. 36

Для следующей автономной системы из двух уравнений строятся две кривые, соответствующие двум начальным условиям, а также поле направлений (рис. 37)

```
> DEplot({diff(x(t),t)=x(t)*(1-y(t)), diff(y(t),
  t)=.3*y(t)*(x(t)-1)},
  [x(t),y(t)],t=-7..7,[[x(0)=1.2,y(0)=1.2],[x(0)=1,
  y(0)=.7]],
  stepsize=.2,title=`Lotka-Volterra model`,
  color=[.3*y(t)*(x(t)-1),x(t)*(1-y(t)),.1],
  linecolor=t/2,arrows=MEDIUM, method=rkf45);
```

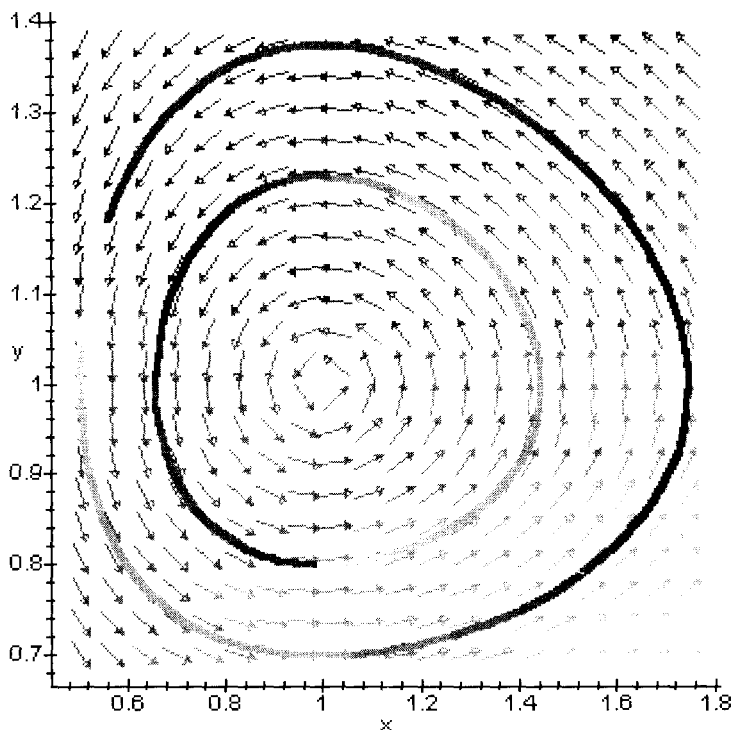

Рис. 37

В пакете **DEtools** имеется также команда **dfieldplot**, непосредственно предназначенная для построения поля направлений системы из двух уравнений первого порядка, а также команда **phaseportrait**, непосредственно предназначенная для построения решений и фазовых портретов систем первого порядка и дифференциальных уравнений более высокого порядка. Впрочем, функции этих команд охватываются командой **DEplot**.

Графика геометрического пакета

В геометрическом пакете построение графических объектов осуществляется при помощи команды

draw(объект1, объект2, ...), где объект — геометрический объект.

Приведем примеры

```
> with(geometry):
```

Определяем треугольник Т

```
Warning, new definition for circle
Warning, new definition for ellipse
Warning, new definition for hyperbola
Warning, new definition for line
Warning, new definition for point
```

```
> triangle(T,[point(A2,0,0),point(A1,2,4),
   point(A3,7,0)]):
```

Находим описанную вокруг треугольника Т окружность

```
> circumcircle(C,T,'centername'=OO):
```

находим высоты Т (altitudes)

```
> altitude(A2A22,A2,T,A22):
  altitude(A3A33,A3,T,A33):
  altitude(A1A11,A1,T,A11):
```

Находим центр вписанной окружности (**orthocenter**) и центр тяжести (**centroid**) треугольника Т

```
> orthocenter(H,T): centroid(G,T):
```

Находим медианы Т

```
> median(A1M1,A1,T,M1):
  median(A2M2,A2,T,M2):
  median(A3M3,A3,T,M3):
> dsegment(dsg1,OO,H): dsegment(dsg2,H,G):
  dsegment(OM1,OO,M1): dsegment(OM2,OO,M2):
  dsegment(OM3,OO,M3):
  triangle(T1,[M1,M2,M3]):
```

Проверяем, лежат ли на одной прямой H, OO, G.

```
> AreCollinear(OO,H,G);
```

true

Выводим на дисплей построенные геометрические объекты (рис. 38)

```
> draw([C(color='COLOR'(RGB,1.0,1.0,.8),filled=true),
  T(color=blue),T1,A3M3,A2M2,A1M1,A2A22,A3A33,A1A11,
  dsg1(style=LINE,color=green,thickness=3),
  dsg2(thickness=3,color=green),
  OM1,OM2,OM3],axes=NONE);
```

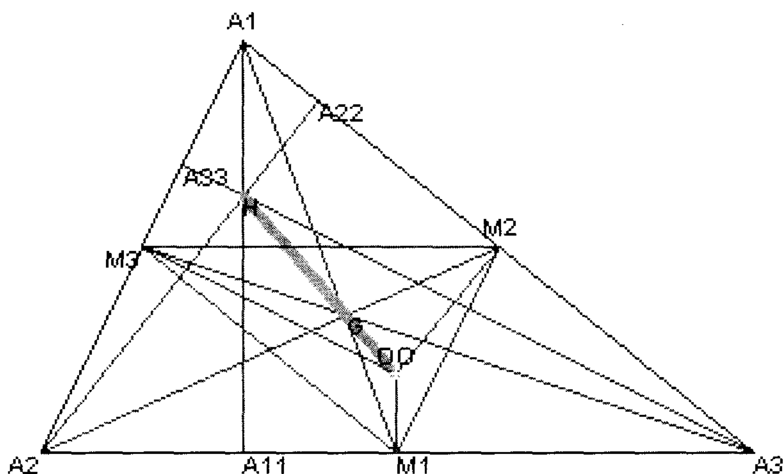

Рис. 38

7.2. Трехмерные графики и трехмерная анимация

Для построения поверхностей в трехмерном пространстве используется команда **plot3d,** а также команды пакетов **plots**, **plottools**, **DEtools**.

- Необязательные дополнительные опции позволяют изменять вид трехмерных графиков:
- опция **grid** позволяет определять размер прямоугольной сетки для меток (значение по умолчанию — 25 × 25);
- при помощи опции **style** можно определять стиль представления поверхности (например **PATCH**, **WIREFRAME**, **POINT**);
- опциями **color** и **shading** задаются различные схемы окраски;
- опции **ambientlight** и **light** позволяют применить освещение рассеянным или направленным светом соответственно;
- опция **orientation** позволит определить точку наблюдения поверхности;
- график можно снабдить заголовком, метками и задать количество делений на осях при помощи опций **title**, **labels**, **tuckmarks** соответственно.

Графики команды *plot3d*

Далее приведены примеры наиболее часто используемых типов трехмерных графиков.

График явно заданной функции (рис. 39)

```
> plot3d(sin( x * y), x=-1.5 ..1.5,
  y=-1.5..1.5, color=WHITE, style=PATCH,
  light=[45,45,1,1,1.4],title='СЕДЛО');
```

СЕДЛО

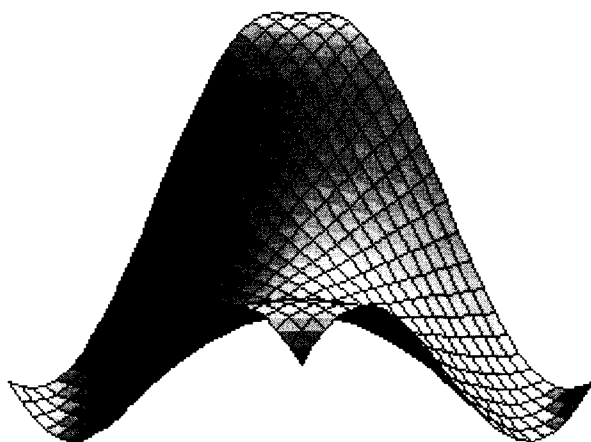

Рис. 39

Можно задавать различные координатные системы (сферическую, тороидальную и так далее — всего тридцать) (рис. 40)

```
> plot3d([x^(1/4)+y^(-1/4),x,y],x=0..2*Pi,
  y=0..2*Pi, coords=toroidal(10));
```

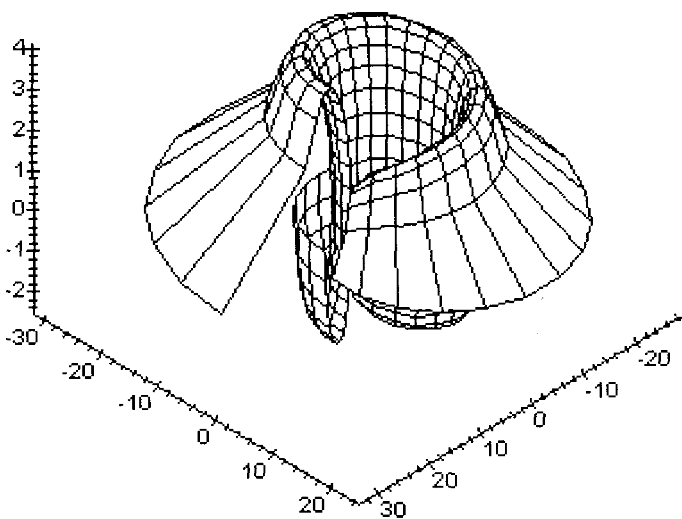

Рис. 40

в некоторых случаях — устанавливать переменные границы диапазона (рис. 41)

```
> plot3d(sin(y*sin(x)),x=-Pi..Pi,y=-x..x);
```

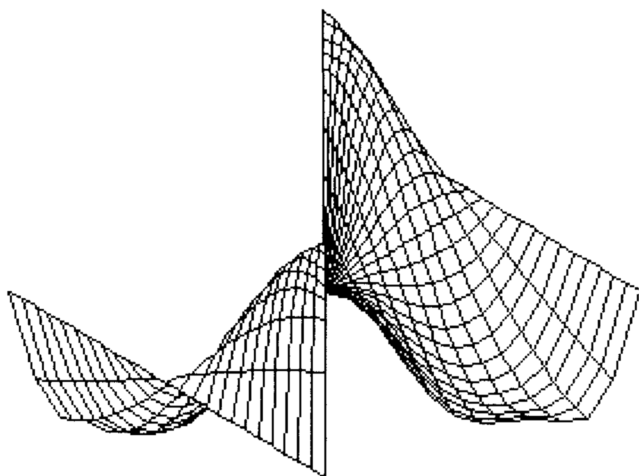

Рис. 41

задать функцию (или процедуру) цвета (рис. 42)

```
> plot3d(x*exp(-x^2-y^2),x=-2..2,y=-2..2,color=x*y);
```

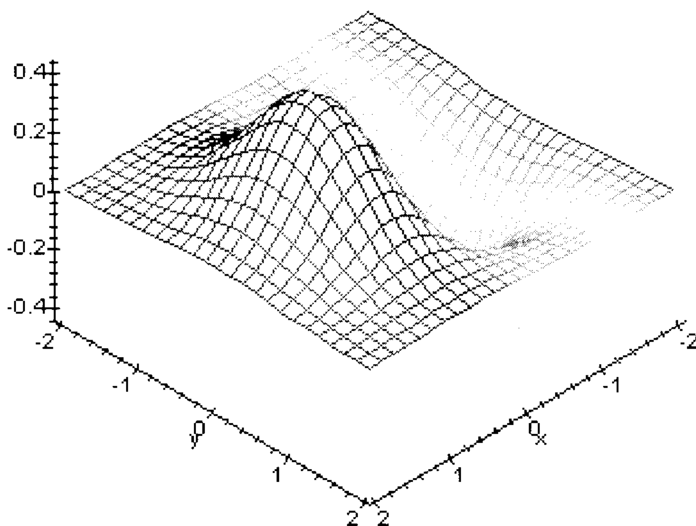

Рис. 42

График поверхности, заданной параметрически тремя функциональными операторами от двух переменных u и v (рис. 43)

```
> Kx:= (u,v) -> 2*(cos(u)     + u*sin(u))*sin(v)/(1 +
  (u*sin(v))^2):
> Ky:= (u,v) -> 2*(sin(u)     - u*cos(u))*sin(v)/(1 +
  (u*sin(v))^2):
> Kz:= (u,v) -> log(tan(v/2)) +         2*cos(v)/(1 +
  (u*sin(v))^2):
> plot3d([Kx,Ky,Kz], -4..4, .01..Pi-.01,
  grid=[35,35]);
```

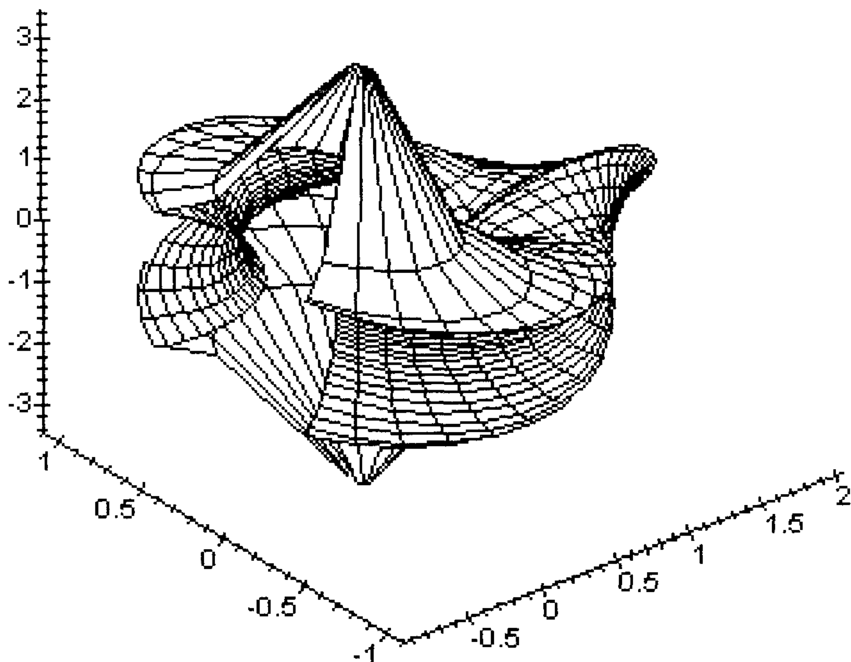

Рис. 43

Другой способ построения поверхности, заданной параметрически тремя функциями от переменных u и t (рис. 44)

```
> plot3d([cos(t)*(1+.2*sin(u)),sin(t)*(1+.2*sin(u)),
  .2*sin(t)*cos(u)],t=0..2*Pi,u=-Pi..Pi);
```

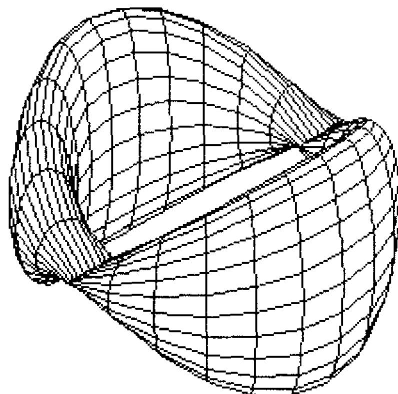

Рис. 44

Несколько поверхностей на одном графике:

```
> plot3d ( { x*sin(y^2), 1-y*cos(x^2) }, x=-1..1,
  y=-1..1 );
```

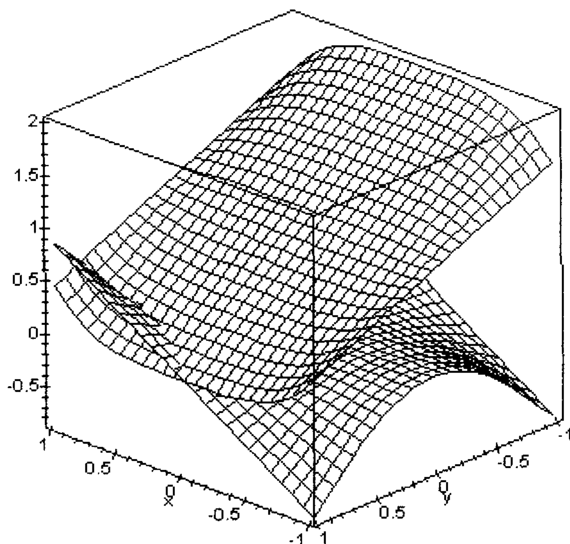

Рис. 45

Построение трехмерных графиков с помощью команд пакета *plots*

Пакет **plots** содержит функцию **coordplots**, предназначенную для построения координатных плоскостей различных систем координат в пространстве. Полное количество систем координат — тридцать, среди них имеются как часто используемые — прямоугольная, сферическая, цилиндрическая, тороидальная, — так и экзотические — шестисферная (**sixsphere**), конфокальная параболическая (**confocalparab**) и другие. Перечислим английские наименования всех систем координат:

bipolarcylindrical, bispherical, cardiodal, cardiodcylindrical, casscylindrical, confocalellip, confocalparab, conical, cylindrical, ellcylindrical, ellipsoidal, hypercylindrical, invcasscylindrical, invellcylindrical, invoblspheroidal, invprospheroidal, logcoshcylindrical, logcylindrical, maxwellcylindrical, oblatespheroidal, paraboloidal, paraboloidal2, paracylindrical, prolatespheroidal, rosecylindrical, sixsphere, spherical, tangentcylindrical, tangentsphere and toroidal.

Приведем примеры (рис. 46—48):

```
> with(plots):
  Digits := 10:
  coordplot3d(spherical);
```

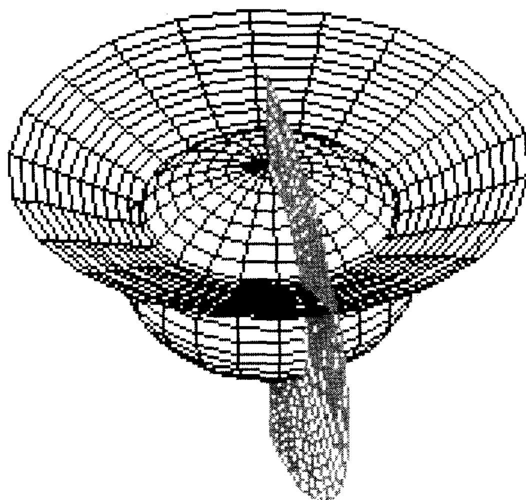

Рис. 46

```
> infolevel[coordplot3d]:=2:
  coordplot3d(rosecylindrical);
```

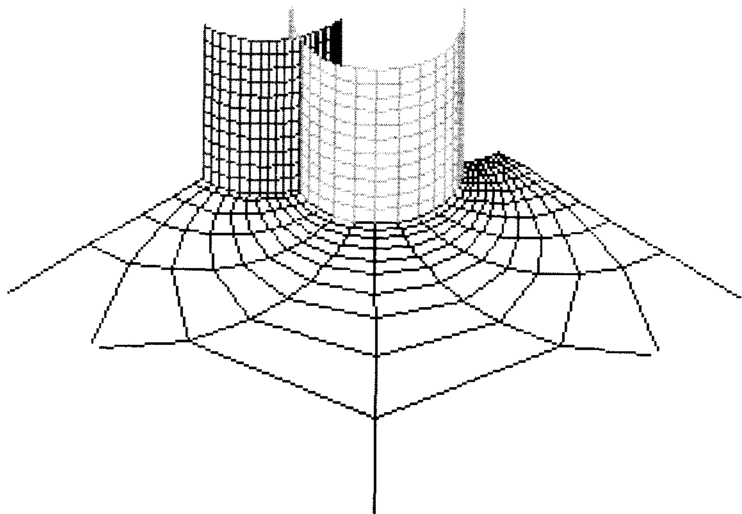

Рис. 47

```
> coordplot3d(sixsphere);
```

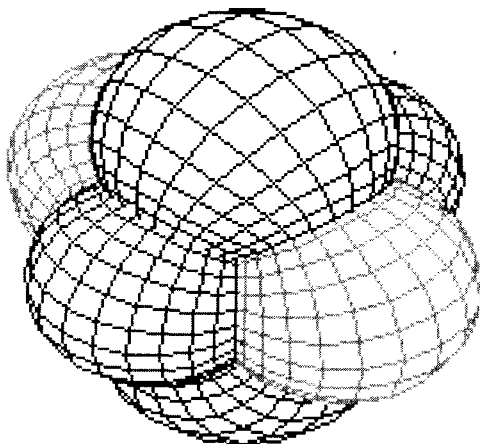

Рис. 48

Команда **cylinderplot** пакета позволяет строить графики в цилиндрических координатах (рис. 49)

```
> f := (5*cos(y)^2 -1)/3;
  plots[cylinderplot](f, x=0..2*Pi,y=-Pi..Pi,
  style=PATCH);
```

$$f := \frac{5}{3}\cos(y)^2 - \frac{1}{3}$$

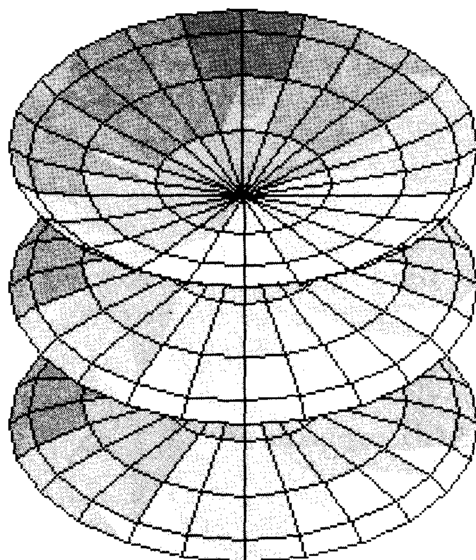

Рис. 49

Команда **complexplot3d** позволяет строить графики комплексных функций в трехмерном пространстве, причем возможны два варианта записи, в виде

```
> complexplot3d( f(z), z = z1..z2 );
```

где f — функция комплексного аргумента z, в этом случае координата z поверхности определяется абсолютной величиной функции, в то время как цвет поверхности определяется аргументом.

В записи

```
> complexplot3d( [f1(x,y), f2(x,y)], x = x1..x2,
  y = y1..y2);
```

координата z определяется функцией f1, а цвет — функцией f2.

Приведем примеры (рис. 50, рис. 51)

```
> complexplot3d( sec(z), z = -3 - 3*I .. 3 + 3*I );
```

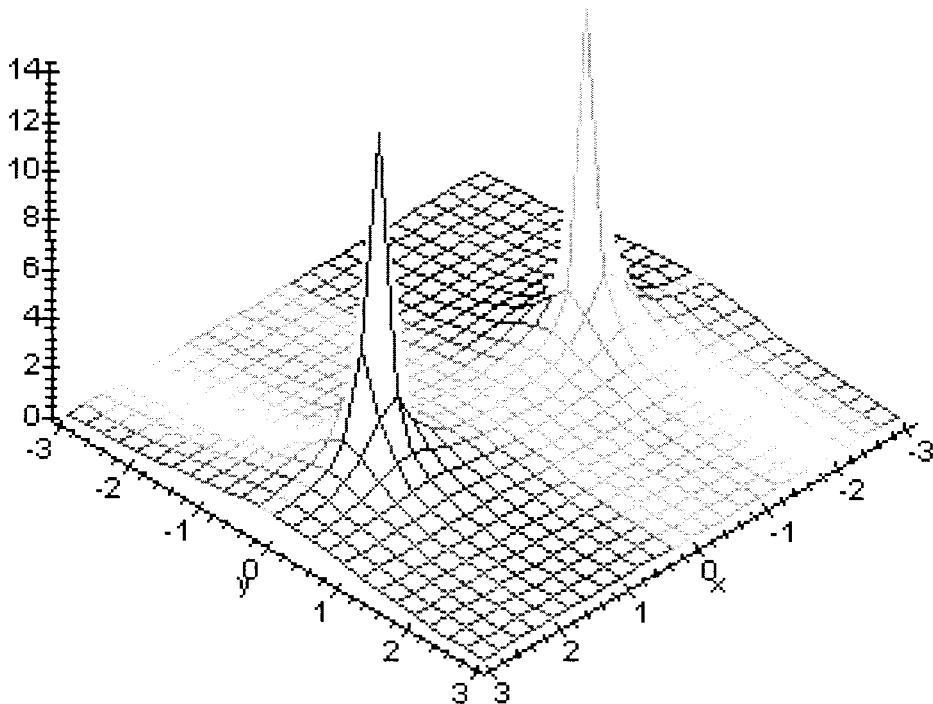

Рис. 50

график преобразования из R2 в R2:

```
> with(plots):complexplot3d( [x^2 - y^2, 2*x*y],
  x = -2..2, y = -2..2);
```

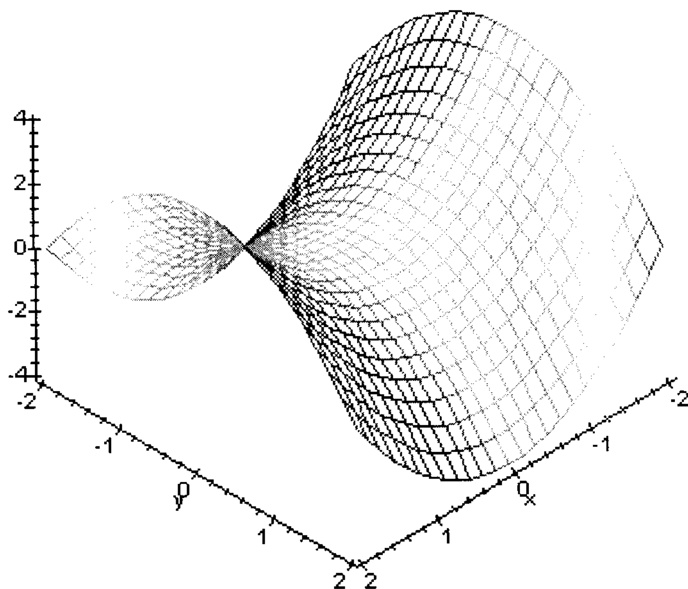

Рис. 51

При помощи команды **fieldplot3d** пакета возможно построение трехмерных векторных полей (рис. 52)

```
> fieldplot3d([2*z*y,2*x*z,2*x*y],x=-1..1,y=-1..1,
  z=0-1..1,grid=[5,5,5]);
```

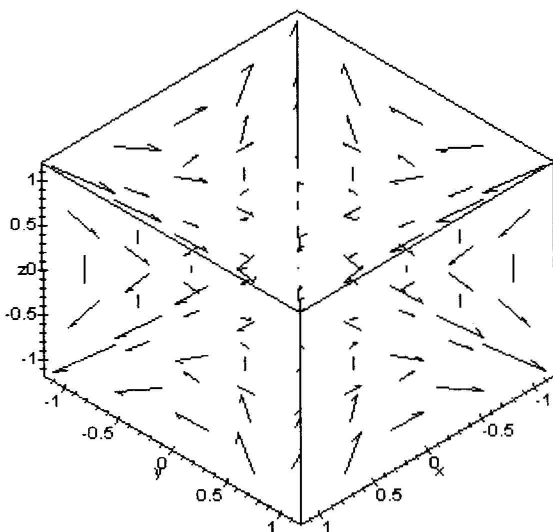

Рис. 52

Команда **gradplot3d** предназначена для построения поля градиентов функции (рис. 53)

```
> gradplot3d( (x^2+y^2+z^2+1)^(1/2),x=-2..2,
  y=-2..2, z=-2..2);
```

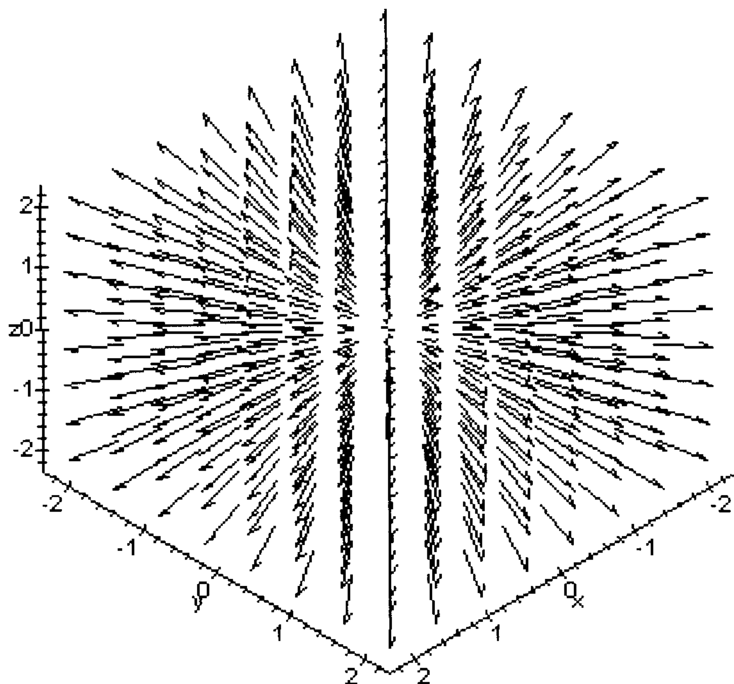

Рис. 53

При помощи команды **implicitplot3d** строятся поверхности, заданные неявно (рис. 54)

```
> implicitplot3d( x^3 + y^3 + z^3 + 1 =
  (x + y + z + 1)^3,x=-2..2,y=-2..2,z=-2..2,
  grid=[13,13,13]);
```

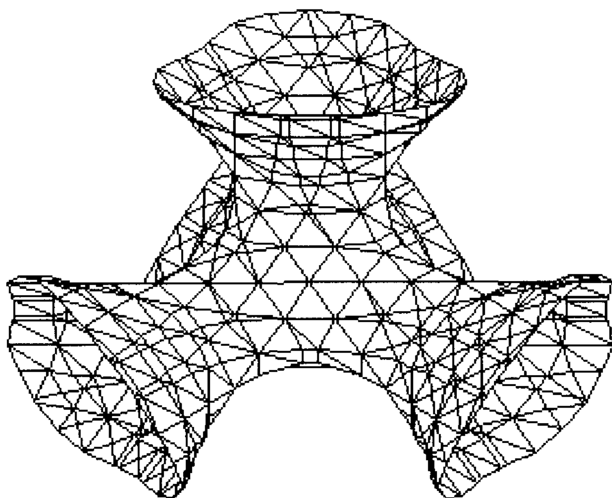

Рис. 54

Команда **listplot3d** предназначена для построения поверхности по точкам, заданным списком списков (матрицей) (рис. 55)

```
> listplot3d([seq([seq(sin((i-15)*(j-10)/Pi/20),
  i=1..30)],j=1..20)]);
```

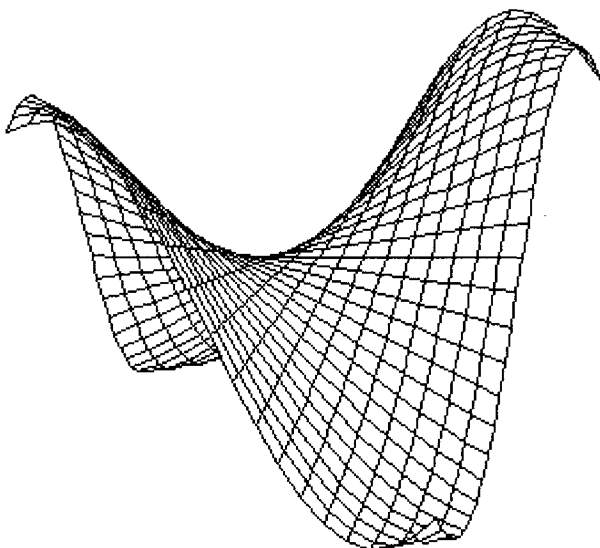

Рис. 55

Команда пакета **matrixplot** строит поверхность, z-координата которой задается матрицей (рис. 56)

```
> with(linalg):
  A:= hilbert(8): B:= toeplitz([1,2,3,4,-4,-3,-2,-1]):
  matrixplot(A+B,heights=histogram,axes=frame,
  gap=0.25, style=patch);
Warning, new definition for norm
Warning, new definition for trase
```

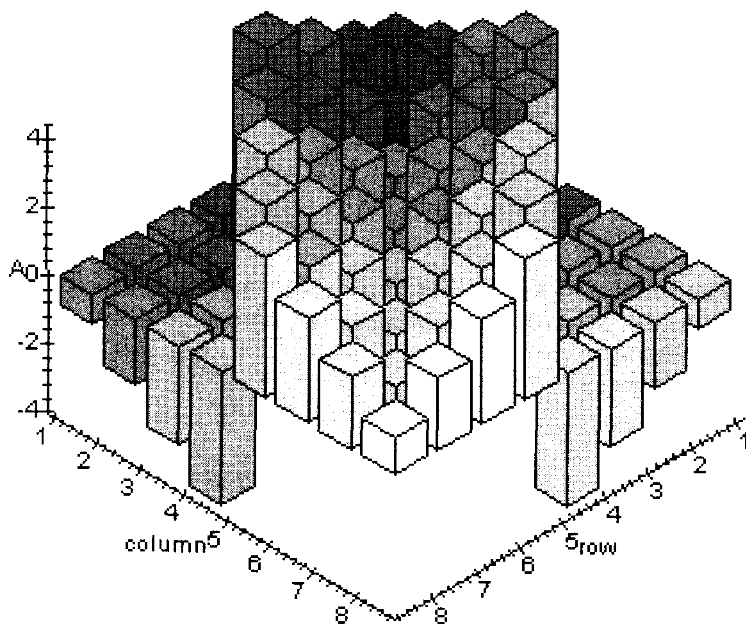

Рис. 56

Команда **odeplot**, входящая в пакет, позволяет строить графики решения дифференциальных уравнений и систем. Рассмотрим систему.

```
> sys := diff(y(x),x)=z(x),diff(z(x),x)=y(x):
  fcns := {y(x), z(x)}:
  p:= dsolve({sys,y(0)=0,z(0)=1},fcns,type=numeric):
```

Трехмерная кривая решения (рис. 57)

```
> odeplot(p, [x,y(x),z(x)],-4..4, numpoints=25,
  color=orange);
```

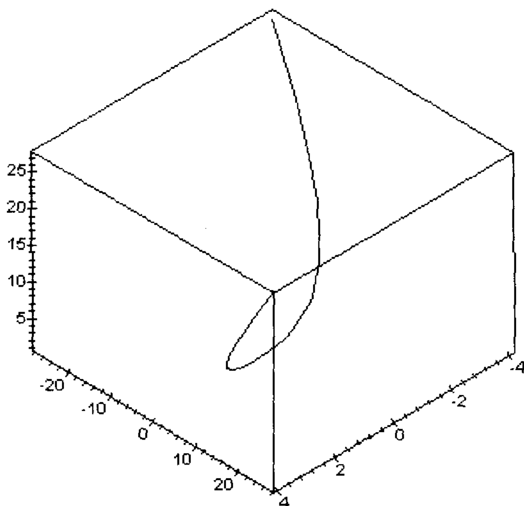

Рис. 57

Команда **polygonplot3d** используется для создания трехмерного графика из многоугольников. Многоугольник задается списком точек, определяющих вершины многоугольника.

Приведем пример (рис. 58)

```
> list_polys :=
  [seq([seq([T/10,S/20,sin(T*S/20)],T=0..20)],
  S=0..10)]:
  polygonplot3d(list_polys);
```

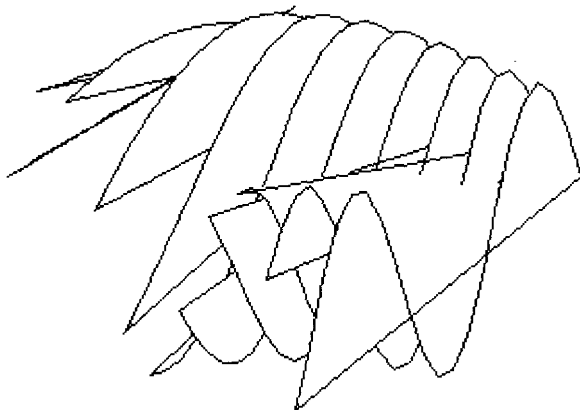

Рис. 58

Команда **spacecurve** предназначена для построения параметрически заданных кривых в пространстве (рис. 59)

```
> curve:= [ -10*cos(t) - 2*cos(5*t) + 15*sin(2*t),
            -15*cos(2*t) + 10*sin(t) - 2*sin(5*t),
            10*cos(3*t),
  t= 0..2*Pi]:
  spacecurve(curve);
```

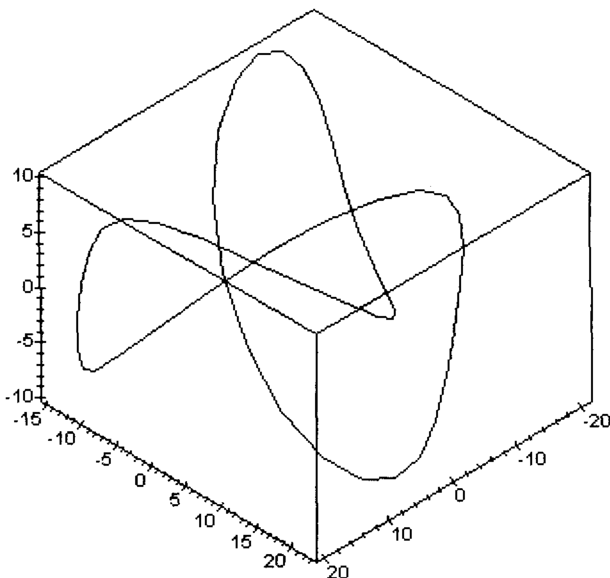

Рис. 59

При помощи команды **surfdata** строится поверхность по заданным точкам (рис. 60)

```
> with(plots):
  data := [seq([ seq([i,j,evalf(cos((i+j)/5))],
  i=-5..5)], j=-5..5)]:
  F := (x,y) -> x^2 + y^2:
  surfdata( data, axes=frame, labels=[x,y,z],
  color=F );
```

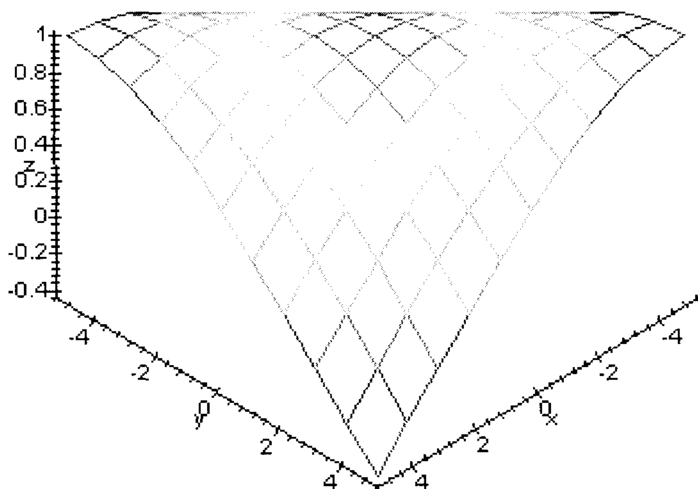

Рис. 60

Команда **textplot3d** позволяет делать надписи на трехмерном графике (рис. 61)

```
> textplot3d([[1,2,3,'The first point'],[2,2,3,
  'Second point']],color=green);
```

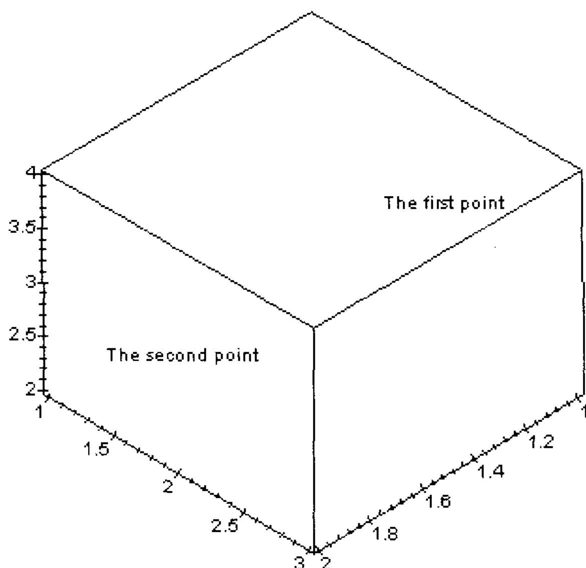

Рис. 61

И, наконец, командой **tubeplot** можно создавать трубчатые графические объекты (рис. 62)

```
> F := (x,y) ->sin(x):
  tubeplot({[cos(t),sin(t),0],[0,sin(t)-1,cos(t)]},
  t=0..2*Pi, radius=t^(1/2)/8,style=patch);
```

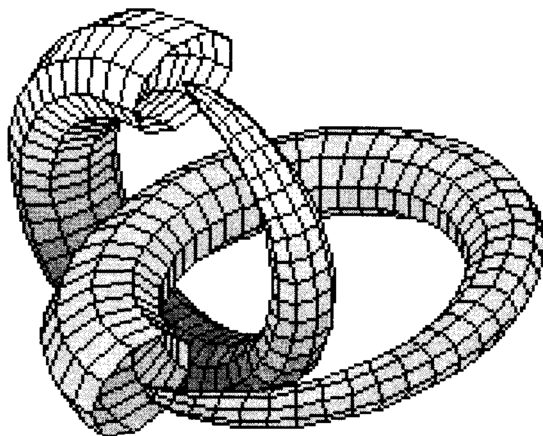

Рис. 62

Графика пакета *DEtools*

Для заданной системы дифференциальных уравнений и списка начальных данных команда **DEplot3d** осуществляет трехмерное представление кривых решения системы. При этом система должна иметь только одну независимую переменную. Поле направлений этой командой (в отличие от команды **DEplot**) не строится.

Приведем пример (рис. 63)

```
> with(DEtools):
> DEplot3d({D(x)(t)=y(t),D(y)(t)=-x(t)-y(t)},[x(t),
  y(t)],t=0..10,
  [[x(0)=0,y(0)=1],[x(0)=0,y(0)=.5]],scene=[t,x(t),
  y(t)], stepsize=.1,
  title=`Damped oscillations`,linecolour=t-sqrt(t));
```

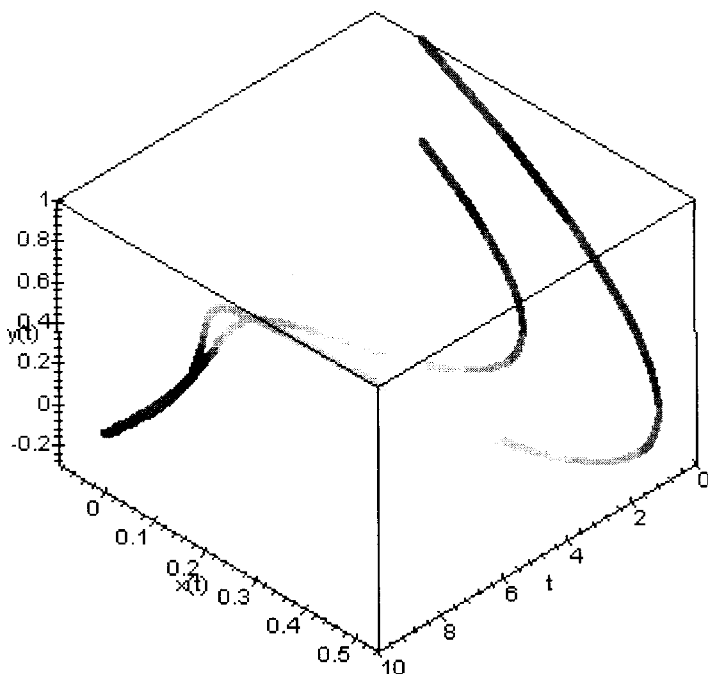

Рис. 63

Команда **PDEplot** пакета позволяет строить графики решений уравнений в частных производных. Эта функция строит поверхность решения квазилинейного уравнения первого порядка вида P(x,y,u) * D[1](u)(x,y) + Q(x,y,u) * D[2](u)(x,y) = R(x,y u), где P, Q и R зависят только от x, y и u(x,y).

Приведем пример (рис. 64).

```
> PDEplot([1,z(x,y),0],z(x,y),[0,s,sech(s)],s=-5..5,
  numsteps=[10,30],numchar=30,
  basechar=true,method=internal,style=HIDDEN,
  orientation=[5,67]);
```

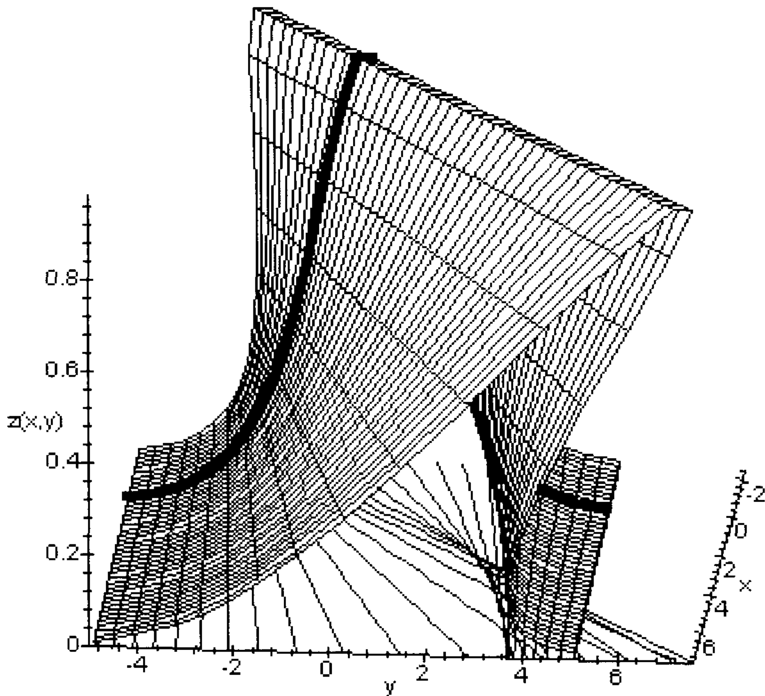

Рис. 64

Графика пакета *plottools*

Пакет **plottools** содержит следующие команды, позволяющие создавать трехмерные графические примитивы, чтобы использовать их в других графиках:

cone, cuboid, cutin, cylinder, dodecahedron, hemisphere, hexahedron, icosahedron, octahedron, semitorus, sphere, tetrahedron, torus.

Приведем примеры (рис. 65)

```
>  with(plottools):
  f := octahedron([0,0,0],0.8), octahedron
  ([1,1,1],0.5):
  plots[display](f,style=patch);
```

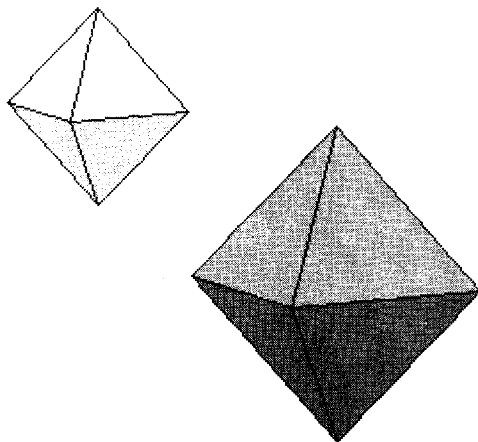

Рис. 65

На следующем примере показано, как можно построить поверхность, состоящую из правильных многогранников, в данном случае — додекаэдров (рис. 66)

```
> data := seq(seq(dodecahedron([i,j,
  evalf(4*cos((i+j)/5))],0.5), i=-5..5), j=-5..5):
> plots[display]( data, axes=frame, labels=[x,y,z]);
```

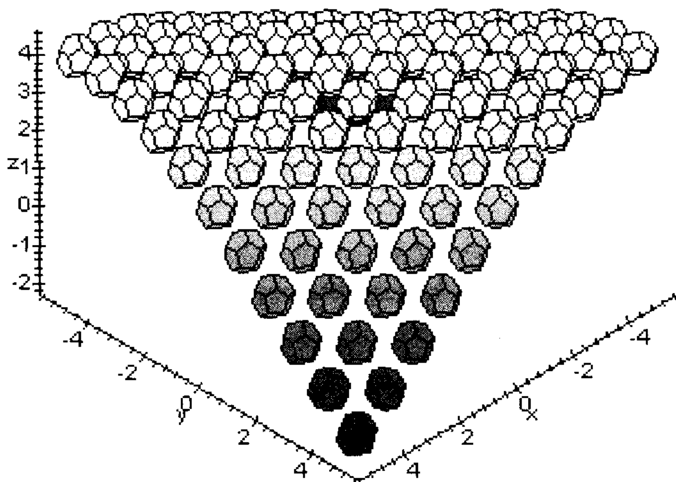

Рис. 66

Трехмерная анимация

Трехмерная анимация выполняется командой **animate3d** из пакета **plots**. Для осуществления анимации в аргумент команды **animate3d** добавлен параметр — переменная анимации и необязательная опция **frames**. Диапазон изменения переменной анимации определяет степень деформации графика, а опция **frames** — число кадров анимации (по умолчанию — 8).

Далее пример трехмерной анимации (рис. 67)

```
> with(plots):
  animate3d(cos(t*x)*sin(t*y),x=-Pi..Pi, y=-Pi..Pi,
  t=1..2);
```

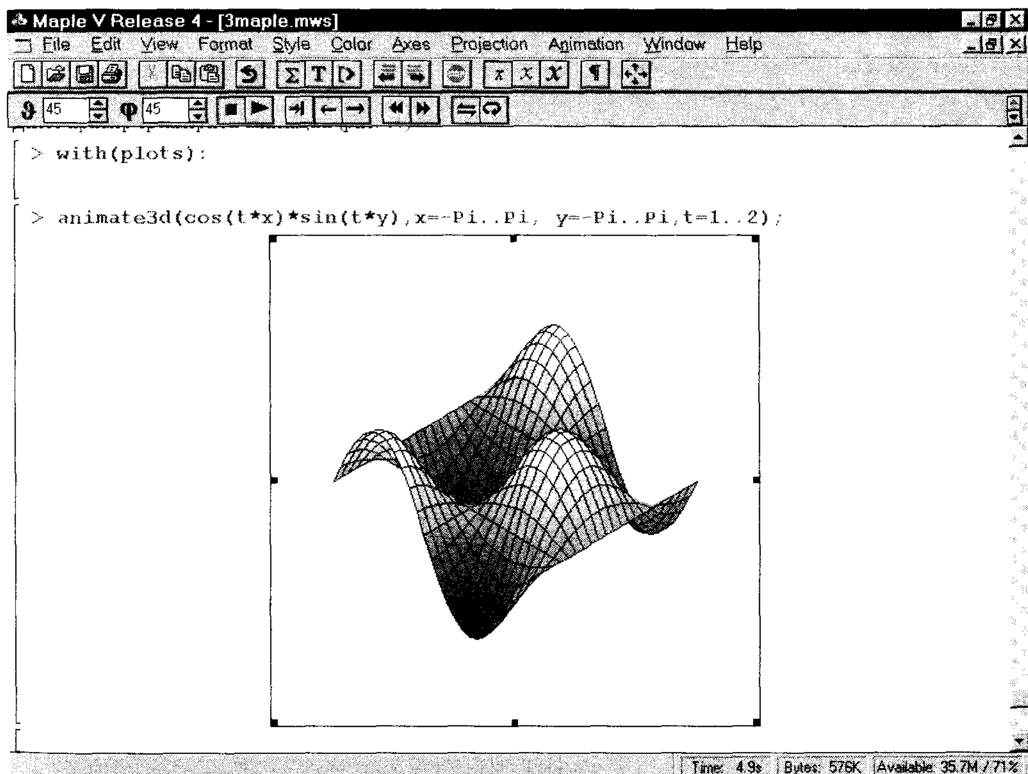

Рис. 67

На этом рисунке вид интерфейса программы *Maple*, когда анимационная картинка активизирована. При активизации картинки анимации на инструментальной панели появляются кнопки управления, сходные с кнопками управления магнитофоном.

8. Программирование в среде *Maple*

Хотя, как уже было показано в предыдущих главах, много полезных и удивительных результатов можно получить, используя интерактивный режим вычислений, по мере освоения программы *Maple* вы все больше будете ощущать необходимость создания программ. Как упоминалось во введении к этому руководству, более чем 90 % встроенных в систему команд запрограммированы на собственном, подобном *Фортрану*, языке программирования *Maple V*, позволяющем создавать программы как символьных, так и численных расчетов. В некоторых случаях обойтись без программирования трудно или вообще невозможно, например при выполнении громоздких повторных вычислений, а также для создания расширений, пополняющих существующую библиотеку.

Эта глава построена следующим образом. В разделе 8.1. будет рассмотрено обычное, процедурного программирования, использующее условные переходы и циклы. В разделе 8.2. — методы пополнения базы данных собственными функциями и операторами, задания свойств этих функций и правил их вычисления. В разделе 8.3 — новый пакет Maple V Domains, средства которого предназначены для ускорения разработки сложных алгоритмов

8.1. Процедурное программирование

8.1.1. Базисные конструкции языка

Базисные конструкции языка программирования *Maple* аналогичны соответствующим конструкциям языков высокого уровня. Перечислим наиболее полезные из них.

If/then/else/fi

При помощи этой конструкции обеспечиваются условные переходы.

Приведем пример.

```
> x:=11;if x < 10 then x^2 else x^3 fi;
```

$$x := 11$$

$$1331$$

Обратите внимание на “fi” (if в обратном направлении) в конце выражения — это признак конца конструкции. После каждой из лексем **then** и **else** не обязательно стоит один оператор, как в приведенном примере, можно вставлять любое количество программных кодов. Обратите внимание также, что команда условного перехода заканчивается знаком конца команды — точкой с запятой “;”. Это не простое совпадение — **if/then/else/fi** — такая же команда *Maple,* как и любая другая команда и должна заканчиваться точкой с запятой.

К конструкции условного перехода может также добавляться элемент **elif**:

If/then/elif/then/.../else/fi

Это команда условного перехода с введением дополнительных условий **elif/then**.

Причем можно устанавливать любое число таких дополнительных условий

```
> if 0<x and x<10 then x^2 elif x<0 then x else x^3 fi;
```

1331

Maple обрабатывает циклы при помощи двух различных конструкций, в одной из которых

for/from/by/to/do/od

для задания цикла используются верхняя и нижняя границы и величина шага для переменной, меняющейся в диапазоне от нижнего к верхнему значению границы. Цикл выполняет команды между **do** и **od** соответствующее число раз. Переменная диапазона может быть также включена в вычисления. Причем между лексемами **do** и **od** может быть включено любое число команд *Maple*. В следующем примере переменная диапазона i меняется от нижней границы 0 до верхней границы 30 с шагом 10 и на каждом шагу выполняется оператор **print(i^2)**:

```
> for i from 0 by  10 to 30 do print(i^2) od;
```

0
100
400
900

Еще одна конструкция для задания цикла

While/do/od

Maple будет повторять оператор, заключенный между **do** и **od**, пока логическое соотношение, записанное между лексемами **while** и **do**, не станет истиной.

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

n := 1

```
> while n < 10 do n:= n^2+1; od;
```

n := 2
n := 5
n := 26

8.1.2. Процедуры

Для того чтобы создать процедуру (подпрограмму), которую вы могли бы использовать неоднократно, в программе *Maple* используется конструкция **proc/end**. Процедура записывается следующим образом

```
> Имя:=proc(параметр1::type1,параметр2::type2, ...)
  local l1,l2...; global g1,g2...; options op1,
  op2, ...; тело процедуры; end;
```

Она начинается с имени, которому присваивается ключевое слово **proc** (сокращенное от procedure), за которым в скобках перечисляются формальные параметры процедуры с необязательным указанием их типа — через дважды записанное двоеточие. Далее может идти необязательное перечисление локальных и глобальных переменных, используемых в теле процедуры, заканчиваемое знаком ";". Вслед за этим, если необходимо, идет перечисление опций процедуры, заканчиваемое знаком ";". Далее идет тело процедуры — алгоритм выполнения процедуры. Процедура обязательно заканчивается словом **end** и следующим за ним знаком конца команды (двоеточие или точка с запятой). Результатом выполнения процедуры является результат последней выполненной операции, если не применены одна из команд возврата **RETURN** или **ERROR** (смотрите ниже).

В следующем примере очень простая процедура **plotdiff** строит кривые функции и ее производной на одном графике (рис. 68).

```
> restart;plotdiff:=proc(y,x,a,b) local yp;
  yp:=diff(y,x);
  plot([y,yp],x=a..b);
  end;
```

plotdiff := **proc**(*y*, *x*, *a*, *b*) **local** *yp*; *yp* := diff(*y*, *x*); plot([*y*, *yp*], *x* = *a* .. *b*) **end**

```
> plotdiff(x^3-2*x+1,x,-1,1);
```

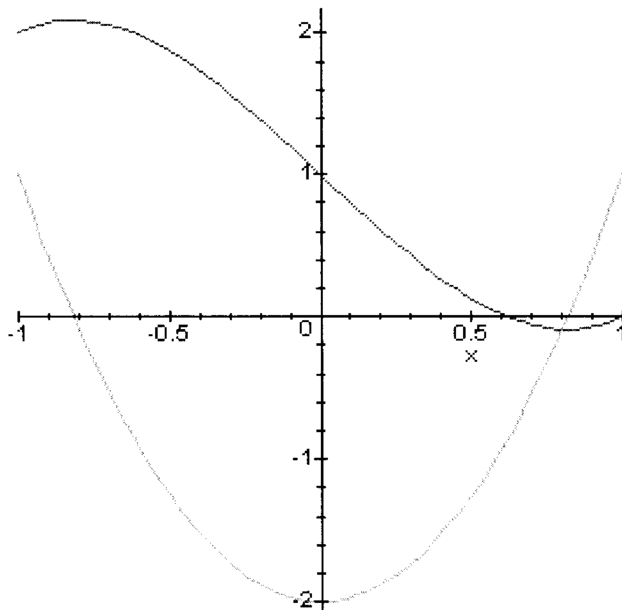

Рис. 68

В этой процедуре значение производной выражения присваивается локальной переменной.

Локальные переменные — это временные ячейки памяти для сохранения значений переменных внутри процедуры, они образуются при каждом вызове процедуры. Правила вычисления локальных переменных внутри процедур отличаются от правил вычисления переменных в командной строке (в интерактивном режиме).

Введем команду присваивания некоторой переменной a значения b

```
> a:=b;
```

$$a := b$$

после этого присвоим переменной b значение c

```
> b:=c;
```

$$b := c$$

А теперь введем

```
> a+1;
```

$$c + 1$$

Таким образом, в интерактивном режиме *Maple* вычисляет все произведенные присвоения переменной а до конца и выводит последнее присвоенное значение. Используя команду **eval** можно вызвать первое присвоенное значение

```
> eval(a,1);
```

$$b$$

второе присвоенное значение

```
> eval(a,2);
```

$$c$$

а также любой другой уровень присвоения. Команда **eval** без указания уровня вычисляет до последнего уровня.

```
> eval(a);
```

$$c$$

Теперь посмотрим, что будет происходить, если переменная является локальной переменной некоторой процедуры. В качестве примера запишем процедуру

```
> f:=proc()
  local a,b;
  a:=b;
  b:=c;
  a+1
  end;
```

$$f := \mathbf{proc}()\ \mathbf{local}\ a,\ b;\ a := b;\ b := c;\ a + 1\ \mathbf{end}$$

```
> f();
```

$$b + 1$$

Такой результат связан с тем, что при вызове процедуры *Maple* вычисляет только первое присвоенное значение локальных переменных. Функция **eval** позволяет вычислить последнее присвоенное значение.

```
> eval(f());
```

$$c + 1$$

Исключение — **ditto**-оператор ("). Он является одной из переменных операционной среды (смотрите ниже), локальным для процедур. При вызове процедуры *Maple* назначает переменной " значение **NULL** (пустое выраже-

ние). В процессе выполнения процедуры *Maple* присваивает переменной " значение последнего выражения, вычисленного до последнего уровня присваивания:

```
> f:=proc()
  local a,b;
  print(`Вначале ["] имеет значение`,["]);
  a:=b;
  b:=c;
  a+1;
  print(`Теперь ["] имеет значение`,["]);
  end;
```

f := **proc**()
local *a*, *b*;
  print(` *Вначале [" ] имеет значение*` , *["]*);
  a := *b*;
  b := *c*;
  a + 1;
  print(` *Теперь [" ] имеет значение*` , *["]*)
end

```
> f();
```

Вначале ["] имеет значение, [ ]
Теперь ["] имеет значение, [*c* + *1*]

Глобальные переменные доступны изнутри любой процедуры и на интерактивном уровне. Таким образом глобальные переменные внутри процедуры вычисляются также, как в интерактивном режиме, то есть до последнего уровня присваивания, кроме тех случаев, когда глобальная переменная является таблицей, массивом или процедурой. В этих последних трех случаях переменная вычисляется до последнего присвоенного имени (last name evaluation).

Параметры процедуры

В строке определения процедуры стоят формальные параметры. Их имена могут быть произвольными (при вызове процедуры используются фактические имена параметров), однако важно их количество и определение типа. Параметров может не быть вовсе, однако и в этом случае в определении процедуры и при вызове ее нужно обязательно записывать пустые скобки. В общем случае, при вызове процедуры число фактических параметров не обязательно должно совпадать с числом формальных параметров. *Maple* выдает ошибку, если параметр пропущен только в случае, если он необходим на данный момент.

Вычисление параметров происходит следующим образом. Фактические параметры вычисляются полностью еще до передачи внутрь процедуры. Внутри процедуры, везде где появляются в выражениях формальные параметры, они заменяются соответствующими значениями фактических параметров.

Для оперирования параметрами процедуры в *Maple* введены специальные функции:

args[i] или **args**[i..j] — последовательность фактических параметров (аргументов), передаваемых процедуре и **nargs** — число параметров, передаваемых процедуре.

При помощи функции **args**[i] можно выделить часть последовательности параметров, передаваемых процедуре, что очень удобно при программировании некоторых процедур, например, в следующей процедуре определяется максимальное из последовательности чисел

```
> maximum := proc () local r, i;
  r := evalf(args[1]);
  for i from 2 to nargs do ifevalf(args[i]) > r
  then r := evalf(args[i]) fi od;
  r end;
  maximum(Pi,exp(1),3);
```

maximum := **proc**()
local *r*, *i*;
r := evalf(args[]);
for *i* **from** 2 **to** nargs **do if** *r* < evalf(args[*i*]) **then** *r* := evalf(args[*i*]) **fi od**;
r
end

$$3.141592654$$

Формальные параметры процедуры можно также применить для передачи внутрь процедуры имени с целью присваивания результата выполнения процедуры. Запишем процедуру

```
> Square:=proc(x::anything, y::name)
  y:=x^2
  end;
```

Square := **proc**(*x*::*anything*, *y*::*name*) *y* := *x*^2 **end**

В этой процедуре результат присваивается второму параметру. Пусть, например, этот параметр ans

```
> Square(d,ans);
```

$$d^2$$

Однако при таком использовании параметра нужно соблюдать осторожность. Если мы не отменим присваивание, то *Maple* сообщит ошибку при повторном вызове процедуры.

```
> Square(d,ans);
Error, Square expects its 2nd argument, y, to be of type
name, but received d^2
```

$$d^2$$

Ошибка связана с тем, что *Maple* внутри процедуры не перевычисляет параметр y и пытается присвоить одно выражение другому. Чтобы этого не происходило, нужно имя фактического параметра, в данном случае ans, заключать в прямые кавычки, то есть писать 'ans'.

А теперь запишем первым параметром процедуры переменную a, которой произведено двухуровневое присвоение

```
> a:=b:
  b:=c:
  Square(a,'ans');
```

$$c^2$$
$$c^2$$

Мы видим, что в этом случае выполняется вычисление до последнего уровня присваивания. Это объясняется тем, что передаваемый внутрь процедуры фактический параметр вычисляется на интерактивном уровне еще до передачи внутрь процедуры.

Переменные операционной среды

Эти переменные могут использоваться в качестве переменных для простых присваиваний внутри процедур. Эти присваивания автоматически отменяются при выходе из процедуры. Значение такой переменной не изменяется внутри всех подпрограмм, вызываемых из данной процедуры, если оно не замещено локально. Другими словами, если в подпрограммах их значения изменились, то нет необходимости их восстанавливать, это произойдет автоматически.

Помимо уже упомянутого **ditto**-оператора (", "", """) *Maple* содержит следующие встроенные переменные операционной среды:

- **Digits** — задает число десятичных знаков в числах с плавающей точкой
- **Normalizer** — используется в степенных рядах для упрощения коэффициентов
- **Testzero** — используется в степенных рядах для выявления деления на нуль
- **mod** — используется в арифметике по модулю **m**
- **printlevel** — используется для задания уровня вложенных подпрограмм, выводимых на дисплей при распечатке программы

Введем, например, процедуру:

```
> t := proc() Digits := Digits + 5; end:
```

Выполнение процедуры дает увеличение нормального значения переменной **Digits** на 5:

```
> t();
```

$$15$$

Однако на интерактивном уровне значение этой переменной автоматически возвращается к исходному нормальному значению:

```
> print(Digits);
```

$$10$$

Пользователь также может вводить переменные операционной среды. Их имя должно начинаться с лексемы **_Env**, за которой может следовать любая последовательность разрешенных для имени символов.

Теперь определим пользовательскую переменную операционной среды **_EnvX** и присвоим ей некоторое значение

```
> _EnvX := x^2+1;
```

$$_EnvX := x^2 + 1$$

Напишем процедуру, переопределяющую **_EnvX**

```
> p := proc() _EnvX := 'polynom' end:
  p();
```

$$polynom$$

Однако на интерактивном уровне значение переменной **_EnvX** не изменилось:

```
> _EnvX;
```

$$x^2 + 1$$

Команда прерывания ERROR

С целью прерывания процедуры и вывода соответствующей ошибки, например при неправильном вводе типа параметра, в процедуре используется команда **ERROR**('строка сообщения').

```
> SUM:=proc(n)
  local i,total;
  if not type(n,integer) then
      ERROR('Вводить можно только целое число');
  fi;
  total:=0;
```

```
for i from 1 to n do
total:=total+i;
od;
total
end;
```

SUM := **proc**(*n*)

local *i*, *total*;

if not type(*n*, *integer*) **then** ERROR(*'Вводить можно только целое число'*) **fi**;

total := 0;

for *i* **to** *n* **do** *total* := *total* + *i* **od**;

end

```
> SUM(a);
```

```
Error, (in SUM) Вводить можно только целое число
```

Рекурсивные процедуры, команда RETURN, опция remember

Эти процедуры содержат обращения к самим себе. Напишем в качестве примера процедуру вычисления чисел Фибоначчи

```
> restart;Fibonacci:=proc(n::nonnegint)
        if n<2 then
                    RETURN(n);
        fi;
                    Fibonacci(n-1)+Fibonacci(n-2)
                 end;
```

Fibonacci :=

proc(*n*::*nonnegint*) **if** *n* < 2 **then** RETURN(*n*) **fi**; Fibonacci(*n* − 1) + Fibonacci(*n* − 2) **end**

В этой процедуре явно определен тип переменной n — **nonnegint** (неотрицательное целое число). При попытке вызвать процедуру с объектом другого типа *Maple* выдаст сообщение об ошибке.

В процедуре применена команда **RETURN**(выражение), прерывающая выполнение команды и возвращающее значение "выражения".

В данной процедуре эта команда применена для возвращения числа n при n < 2. Вычислим последовательность чисел Фибоначчи от 1 до 15:

```
> seq(Fibonacci(i),i=0..15);
```

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610

С целью увеличения скорости выполнения расчетов в рекурсивных процедурах вводится опция **remember**, при помощи которой сохраняется в памяти предыдущий результат вычисления. С применением этой опции процедура вычисления чисел Фибоначчи запишется в следующем виде:

```
> F:=proc(n::nonnegint)
  option remember;
  if n<2 then
  RETURN(n)
  fi;
  if irem(n,2)=0 then
  F(n):=2*F(n/2-1)*F(n/2)+F(n/2)^2 else
  F(n):=F((n-1)/2+1)^2+F((n-1)/2)^2 fi
  end;
```

F := **proc**(*n*::*nonnegint*)

option *remember*;

if *n* < 2 **then** RETURN(*n*) **fi**;

if irem(*n*, 2) = 0 **then** F(*n*) := 2*F(1/2**n* − 1)*F(1/2**n*) + F(1/2**n*)^2

else F(*n*) := F(1/2**n* + 1/2)^2 + F(1/2*n − 1/2)^2

fi

end

Если в первом варианте процедуры время вычисления каждого следующего числа Фибоначчи увеличивается по экспоненте, то во втором — линейно с увеличением n. Так, например, 2000-е число Фибоначчи вычисляется почти мгновенно:

```
> F(2000);
```

```
422469633339230487870672560234148278257985284025068109801028013731\
  430858437013070722412359963914151108844608753890960360764019471\
  164359602927198331259873732625355580260699158591522949245390499\
  872225679531698287448247299226390183371677806060701161549788671\
  987985831146887087626459736908672288402365442229524334796448013\
  951534956297208765265606952980649984197744872015561280266540455\
  41717178819303240252043120825 16817125
```

В то же время для первого варианта процедуры уже 20-е число Фибоначчи вычисляется несколько секунд. Сравним при помощи команды **time** времена, затрачиваемые процедурами для вычисления 20-ого числа Фибоначчи:

```
> time(F(20));time(Fibonacci(20));
```

0

6.038

Вложенные процедуры

Вспомним команду **map**. Как вы уже знаете, эта команда применяет функцию к списку.

Если, например, задан список

```
> restart;lst:=[2,4,5,6];
```

$$lst := [2, 4, 5, 6]$$

и мы хотим все его элементы возвести в степень, равную первому элементу списка (2), то можно записать процедуру

```
> map(x->x^lst[1],lst);
```

$$[4, 16, 25, 36]$$

Как записать эту процедуру внутри другой процедуры, которая будет внешней для процедуры **map** (назовем ее out)? Попробуем вначале записать так

```
> v:='v';out:=proc(x::list)
  local v;
  v:=x[1];
  map(y->y^v,x)
  end;
```

$$v := v$$

out := **proc**(*x*::*list*) **local** *v*; *v* := *x*[1]; map(*y* −> *y*^*v*, *x*) **end**

```
> out(lst);
```

$$[2^v, 4^v, 5^v, 6^v]$$

Такая запись процедуры не привела к нужному результату, так как будучи локальной для процедуры out, переменная v не перенесла присвоенное ей значение **x**[1] во внутреннюю процедуру y −> y^v.

Теперь попробуем в процедуре out1 переменную v продекларировать как глобальную

```
> out1:=proc(x::list)
  global v;
  v:=x[1];
  map(y->y^v,x)
  end;
```

out1 := **proc**(*x*::*list*) **global** *v*; *v* := *x*[1]; map(*y* −> *y*^*v*, *x*) **end**

```
> out1(lst);
```

$$[4, 16, 25, 36]$$

Теперь все в порядке, однако продекларируем v как локальную в процедуре out2 и как глобальную во вложенной процедуре

```
> v:='v';out2:=proc(x::list)
  local v;
  v:=x[1];
  map(proc(y) global v; y^v; end,x)
  end;
```

$$v := v$$

out2 := **proc**(*x*::*list*) **local** *v*; *v* := x[1]; map(**proc**(*y*) **global** *v*; *y*^*v* **end**, x) *end*

```
> out2(lst);
```

$$[2^v, 4^v, 5^v, 6^v]$$

Такая запись также не приводит к нужному результату. Это можно объяснить тем, что во внутренней процедуре переменная v является глобальной и поэтому отличается от локальной переменной v процедуры out2.

Попробуем передать переменную v во внутреннюю процедуру через параметр внутренней процедуры

```
> out3:=proc(x::list)
   map(proc(y,z) y^z; end,x,x[1])
   end;
```

out3 := **proc**(*x*::*list*) map(**proc**(*y*, *z*) *y*^*z* **end**, *x*, *x*[1]) **end**

```
> out3(lst);
```

$$[4, 16, 25, 36]$$

Теперь получился правильный результат. Однако есть и другие приемы. Можно использовать команду **unapply** вместо оператора стрелки или **proc()** для создания вложенной процедуры.

```
> unapply(y^v,y);
```

$$y \to y^v$$

Эта команда создает процедуру из математических выражения с аргументом y. Причем, внутри созданной таким образом процедуры переменные, не являющиеся аргументами процедуры такие же какими они являлись вне процедуры, то есть глобальные для процедуры out. Запишем еще один вариант вложенной процедуры.

```
> out4:=proc(x::list)
  unapply(y^x[1],y);
  map(",x)
  end;
```

out4 := **proc**(*x*::*list*) unapply(y^x[1], y); map(", x) **end**

```
> out4(lst);
```

[4, 16, 25, 36]

Еще один прием — использование команды подстановки **subs** для замены глобальной переменной вложенной процедуры значением локальной переменной процедуры out.

```
> out5:=proc(x::list)
  local v;
  global w;
  print('это w',w);
  v:=x[1];
  map(subs('w'=v,y->y^w),x)
  end;
```

out5 := **proc**(*x*::*list*)

local *v*;

global *w*;

print('*это w*', *w*); *v* := x[1]; map(subs('*w*'= *v*, *y* -> *y*^*w*), *x*)

end

```
> out5(lst);
```

это w, w

[4, 16, 25, 36]

Здесь w берется в кавычки, чтобы исключить ошибки в случаях, когда глобальному имени w присвоено некоторое значение. Когда w в кавычках, то используется только имя w. Этот метод передачи значений локальных переменных во вложенную процедуру работает всегда, но программы получаются менее ясными для понимания и более сложными для прочтения.

Итак, подведем итоги. Для передачи значения некоторой переменной a в процедуру int(), вложенную в процедуру out() можно применить следующие приемы:

- в процедуре out() определить переменную a как глобальную;
- передать переменную a во вложенную процедуру int() через параметр вложенной процедуры: int(..,a);

- определить (если это возможно) вложенную процедуру при помощи оператора–функции, созданного командой unapply;
- передать значение локальной переменной a процедуры out() во вложенную процедуру int() при помощи вспомогательной глобальной переменной w процедуры out() и функции подстановки subs(w=a,int()).

Ньютоновская итерация

Как известно итерационный метод дает возможность получать последовательные все более точные значения искомой величины по ее предыдущему значению. Один из наиболее известных итерационных методов — метод касательных Ньютона для нахождения корней алгебраических функций.

В качестве примера символьного программирования напишем процедуру NutonIteration, дающую по заданной функции формулу итерации. Формальными параметрами процедуры будут алгебраическое выражение и имя переменной. Результатом процедуры должна быть функция, выражающая формулу, по которой будут вычисляться последовательные значения переменной.

```
> restart; NewtonIteration:= proc(expr::algebraic,
  x:: name)
  local iteration;
  iteration:=x-expr/diff(expr,x);
  unapply(iteration,x)
  end;
```

NewtonIteration := **proc**(*expr*::*algebraic*, *x*::*name*)
local *iteration*;
iteration := *x* − *expr*/diff(*expr*, *x*); unapply(*iteration*, *x*)
end

Чтобы получить функцию по заданному выражению в процедуре применена команда **unapply**.

В качестве примера возьмем выражение

```
> expr:=x*sin(x)-sqrt(x);
```

$$expr := x\sin(x) - \sqrt{x}$$

График его изображен на рис. 69

```
> plot(expr,x=-1..10);
```

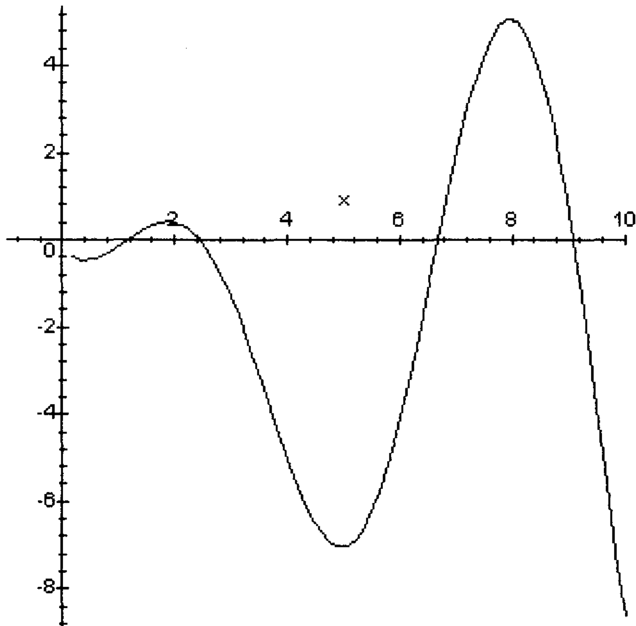

Рис. 69

Применим процедуру

```
> Form:=NewtonIteration(expr,x);
```

$$Form := x \to x - \frac{x\sin(x) - \sqrt{x}}{\sin(x) + x\cos(x) - \frac{1}{2}\frac{1}{\sqrt{x}}}$$

Зададим начальное значение переменной

```
> x0:=9.0;
```

$$x0 := 9.0$$

и найдем несколько Ньютоновских итераций

```
> to 4 do x0:=Form(x0);od;
```

$$x0 := 9.089137810$$
$$x0 := 9.086631757$$
$$x0 := 9.086629934$$
$$x0 := 9.086629934$$

Мы видим, что значение x0 очень быстро приближается к значению корня функции.

Можно эту же процедуру записать в виде, когда формальным параметром процедуры является также процедура

```
> restart;NewtonIteration:=proc(f::procedure)
  (x->x)-f/D(f);
  end;
```

$$NewtonIteration := \mathbf{proc}(f::procedure)\ (x \to x) - f/\mathrm{D}(f)\ \mathbf{end}$$

```
> g:= x-> x*sin(x)-sqrt(x);
```

$$g := x \to x\sin(x) - \sqrt{x}$$

```
> Form2:=NewtonIteration(g);
```

$$Form2 := x \to x - \frac{g}{x \to \sin(x) + x\cos(x) - \frac{1}{2}\frac{1}{\sqrt{x}}}$$

```
> x0:=9.0;
```

$$x0 := 9.0$$

```
> to 4 do x0:=Form2(x0) od;
```

$$x0 := 9.089137810$$
$$x0 := 9.086631757$$
$$x0 := 9.086629934$$
$$x0 := 9.086629934$$

Оператор аффинного преобразования

В качестве еще одного примера, позволяющего освоить приемы программирования в *Maple*, напишем процедуру, вычисляющую по заданной функции F(x) функцию F(a*x+b), — так называемый оператор аффинного преобразования переменной (назовем его aff-оператор).

```
> aff:=proc(f::procedure,a::numeric,b::numeric)
  local x;
  unapply(f(a*x+b),x)
  end;
```

$$aff := \mathbf{proc}(f::procedure,\ a::numeric,\ b::numeric)\ \mathbf{local}\ x;\ \mathrm{unapply}(f(a*x+b),\ x)\ \mathbf{end}$$

Проверим

```
> aff(sin,2,1);
```

$$x \to \sin(2\,x + 1)$$

Построим график (рис. 70)

```
> plot(",0..Pi);
```

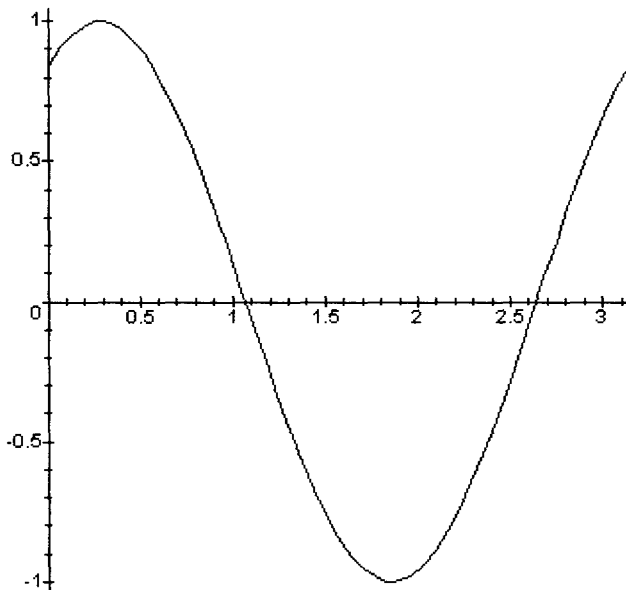

Рис. 70

Мы видим, что процедура работает. Однако возьмем функцию от двух переменных

```
> A:=(x,y)->x*sin(y);
```

$$A := (x, y) \to x \sin(y)$$

```
> aff(A,2,1);
Error, (in aff) aff uses a 4th argument, b (of type numeric),
which is missing
```

Теперь у нас ошибка, поскольку в записи процедуры мы указали только одну переменную для формального параметра. Используем такой прием. Внутри процедуры aff введем процедуру преобразования одного из аргументов

вспомогательной глобальной функции F с произвольным числом аргументов. А затем применим оператор подстановки для замены вспомогательной функции на фактическую. Теперь процедура будет выглядеть так

```
> aff:=proc(f::procedure,a::numeric,b::numeric)
  global F, g, h;
  subs({'F'=f,g=a,h=b},x->F(g*x+h,args[2..-1]))
  end;
```

aff := **proc**(*f*::*procedure*, *a*::*numeric*, *b*::*numeric*)
global *F*, *g*, *h*;
subs({'*F*' = *f*, *h* = *b*, *g* = *a*}, *x* → F(*g***x* + *h*, args[2 .. −1]))
end

Здесь **args**[2..−1] перечисляет все аргументы функции, начиная со второго.

```
> aff(sin,2,1);
```

$$x \to \sin(2\,x + 1, \text{args}_{2\,..\,-1})$$

```
> "(x);
```

$$\sin(2\,x + 1)$$

Теперь проверим, как работает эта программа для функции от двух переменных

```
> Aa:=aff(A,2,1);
```

$$Aa := x \to \text{A}(2\,x + 1, \text{args}_{2\,..\,-1})$$

```
> Aa(x,y);
```

$$(2\,x + 1)\sin(y)$$

Теперь процедура работает для функций от нескольких переменных. Обобщим далее оператор сдвига, чтобы он мог преобразовывать любую переменную функции. Вторым параметром процедуры теперь будет номер сдвигаемой переменной, а третьим и четвертым — параметры преобразования.

```
> aff:=proc(f::procedure,n::posint,a::numeric,
  b::numeric)
  global F,m,g,h;
  subs(F'=f,m=n,g=a,h=b,proc() F(args [1..m-1],
  g*args[m]+h,args[m+1..-1]) end)
  end;
```

aff := **proc**(*f*::*procedure*, *n*::*posint*, *a*::*numeric*, *b*::*numeric*)
global *F*, *m*, *g*, *h*;

subs($'F' = f$, $m = n$, $g = a$, $h = b$,

proc()F(args[1 .. $m - 1$], g*args[m] + h, args[$m + 1$ .. −1]) **end**)

end

Проверим написанную процедуру для функции от двух переменных

```
> Aa2:=aff(A,2,2,1);
```

Aa2 := **proc**() A(args[1 .. 1], 2*args[2] + 1, args[3 .. −1]) **end**

```
> Aa2(x,y);
```

$$x \sin(2\, y + 1)$$

Построим график (рис. 71)

```
> plot({A(1,y),Aa2(1,y)},y=0..10);
```

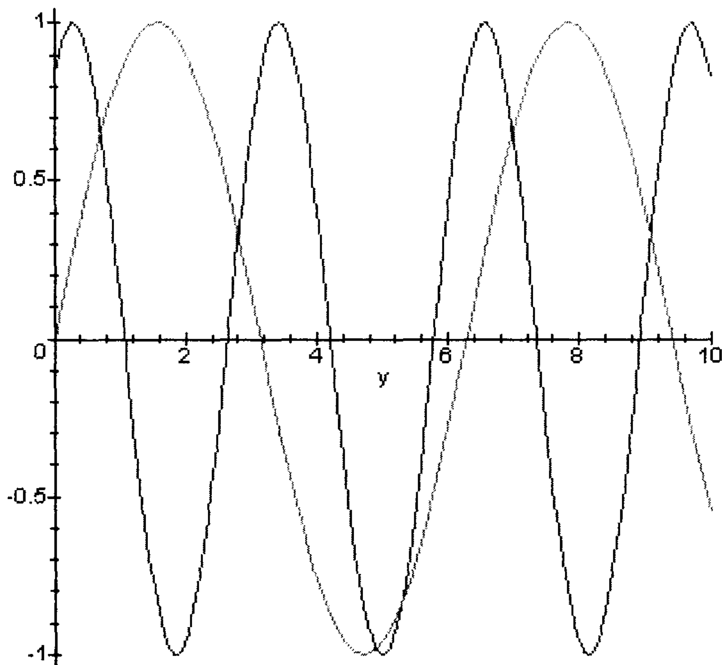

Рис. 71

8.1.3. Методы отладки программ

В этом разделе мы ознакомимся с методами отладки процедур *Maple*.

Трассировка

Наиболее простой метод отладки — применение средства **printlevel**, позволяющего проследить за выполнением программы. **printlevel** — глобальная переменная, которая при начальной загрузке *Maple* равна 1. Если присвоить ей большее целое значение, то при выполнении процедуры на экран будут выводиться последовательность присваиваний, вход и выход из процедуры с перечислением значений параметров процедуры (трассировка процедуры). Величина переменной **printlevel** определяет уровень (глубину) вложенных процедур, которые будут трассироваться при выполнении программы. Для того чтобы читалась процедура с глубиной вложения n нужно задать **printlevel**:=n×5. Применим трассировку для выяснения ошибки в записанной выше процедуре out.

```
> out:=proc(x::list)
  local v;
  v:=x[1];
  map(y->y^v,x)
  end;
```

$$out := \mathbf{proc}(x::list)\ \mathbf{local}\ v;\ v := x[1];\ \mathrm{map}(y \to y\verb|^|v, x)\ \mathbf{end}$$

Зададим для переменной **printlevel** значение, достаточное, чтобы читались процедуры второго уровня вложения.

```
> printlevel:=10;
```

$$printlevel := 10$$

```
> out([2,3,4,5]);
```

```
{--> enter out, args = [2, 3, 4, 5]
```

$$v := 2$$

```
{--> enter unknown, args = 2
```

$$2^v$$

```
<-- exit unknown (now in out) = 2^v}
{--> enter unknown, args = 3
```

$$3^v$$

```
<-- exit unknown (now in out) = 3^v}
{--> enter unknown, args = 4
```

$$4^v$$

```
<- exit unknown (now in out) = 4^v}
{-> enter unknown, args = 5
```

$$5^v$$

```
<- exit unknown (now in out) = 5^v}
```

$$[2^v, 3^v, 4^v, 5^v,]$$

```
<- exit out (now at top level) = [2^v, 3^v, 4^v, 5^v]}
```

$$[2^v, 3^v, 4^v, 5^v,]$$

Из распечатки видно, что во вложенную процедуру не передается значение переменной v. Если в программе встречается системная ошибка, то при значении **printlevel**>3 *Maple* распечатает на экране параметры всех процедур, выполняемых на данный момент, значения локальных переменных и команду, которая выполнялась при возникновении ошибки.

Приведем пример с ошибкой "деление на ноль".

```
> f:= proc(x) local y; z:=5; g(x,y); end;
Warning, 'z' is implicitly declared local
```

$$f := \mathbf{proc}(x) \text{ local } y, z; z := 5; \text{g}(x, y) \text{ end}$$

```
> g:=proc(z,t) local u,v; u:=0; v:=t/u; u+v;end;
```

$$g := \mathbf{proc}(z, t) \mathbf{\ local\ } u, v; u := 0; v := t/u; u + v \mathbf{\ end}$$

Присвоим переменной **printlevel** значение 4.

```
> printlevel:=4;
```

$$printlevel := 4$$

```
> f(x);
{-> enter f, args = x
```

$$z := 5$$

```
<- ERROR in f (now at top level) = division by zero}
Error, (in g) division by zero
 executing statement: v := t/u
 locals defined as: u = 0, v = v
 g called with arguments: x, y
 f called with arguments: x
```

Для трассировки процедур используются также команды *Maple* **trace**, **untrace**, которые, в отличие от вышеописанного метода, позволяют выборочно включать и отключать трассировку некоторых из вложенных процедур, чтобы исключить громоздкий вывод на дисплей.

Эти команды могут использоваться в следующих вариантах:

- **trace**(f);
- **trace**(f,g,h,...);
- **untrace**(f);
- **untrace**(f,g,h,...);

где f, g, h, ... — имена процедур, которые будут трассироваться командой.

Команда **trace** инициирует в течение выполнения программы вывод на дисплей точек входа, результат выполнения операторов программы и точек выхода трассируемой процедуры. В точках входа выводятся значения фактических параметров процедур, в точках выхода — значения возвращаемых функций.

Функция **untrace** выключает трассировку по указанным процедурам.

В качестве примера включим трассировку процедуры g, предварительно установив

```
> printlevel:=1;
```

$$printlevel := 1$$

```
> trace(g);
```

$$g$$

```
> f(x);
{--> enter g, args = x, y
```

$$u := 0$$

```
<-- ERROR in g (now in f) = division by zero}
Error, (in g) division by zero
```

Мы видим, что трассировка при выполнении процедуры f включена только для вложенной процедуры g. Если выполняемая процедура будет вводиться с символом двоеточия в конце команды, то на дисплей будут выводиться только параметры в точках входа и выхода процедуры, а результат выполнения операторов выводиться не будет.

```
> trace(f,g);f(x):
```

$$f, g$$

```
{--> enter f, args = x
{--> enter g, args = x, y
<-- ERROR in g (now in f) = division by zero}
<-- ERROR in f (now at top level) = division by zero}
Error, (in g) division by zero
```

Отладчик

Для отладки процедур в *Maple* имеется также более мощное средство — отладчик процедур. Отладчик вызывается автоматически при встрече с одной из меток, установленных в процедуре: **stopat, stopwhen, stoperror**.

Когда отладчик вызывается, он выводит на экран выражение, команду, которая должна выполняться следующей и приглашение ввода команды отладчика DBG>.

Если отладчик вызван командой прерывания (**stopat**), выводимое на экран выражение есть результат последней выполненной команды. Если он вызван командой наблюдения за переменной (**stopwhen**), выражение представляет собой равенство, левая часть которого — имя наблюдаемой переменной, а правая часть — значение, присвоенное этой переменной. Если отладчик вызван меткой сообщения об ошибке (**stoperror**), выражение — сообщение об ошибке.

Перечислим некоторые команды, которые можно вводить в режиме отладки:

- **cont** — продолжить выполнения процедуры до следующей точки прерывания или до конца;
- **step** — выполнить следующую команду;
- **quit**, **done**, или **stop** — полностью остановить выполнение процедуры;
- **showstat** [имя процедуры] [номер оператора[..номер оператора]] — вывести на экран операторы заданной процедуры с заданными номерами;
- **showstop** — вывести на экран список всех процедур, содержащих метки;
- **stopat** [имя процедуры] [номер оператора] [условие] — установить точку прерывания в заданной процедуре на операторе с заданным номером. Необязательное "условие", которое должно быть булевым выражением, приводит к остановке программы только в случае выполнения "условия";
- **unstopat** [имя процедуры] [номер оператора] — удалить из процедуры указанную точку прерывания;
- **stopwhen** [имя процедуры переменная] — установить точку наблюдения за заданной локальной или глобальной переменной и вывести на экран список точек наблюдения;
- **unstopwhen** [имя процедуры переменная] — удалить из процедуры точки наблюдения за заданной переменной;
- **stoperror** [сообщение об ошибке] — установить точку наблюдения за заданной ошибкой или вывести список установленных меток;
- **unstoperror** [сообщение об ошибке] — удалить точку наблюдения за ошибкой;
- **любое выражение** ***Maple*** — вычислить значение выражения в точке останова.

Приведем пример. Пусть задана процедура и команда прерывания, включающая отладчик,

```
> f := proc(x,y) local a; global b;
        if x < y then
            a := x; b := y + a;
        else
            a := y; b := x + a;
        fi;
        a + b + x + y
  end:
  stopat(f);
```

$$[f]$$

```
> f(2,3);
f:
   1*   if x < y then
           ...
        else
           ...
        fi;

> stopwhen b # `устанавливаем точку наблюдения за
  переменной b`
[b]
f:
   1*   if x < y then
           ...
        else
           ...
        fi;
> cont          # `продолжаем выполнение программы`
b := 5
f:
   6    a+b+x+y
> showstat      # `выводим на экран текущее состояние`
f := proc(x, y)
local a;
global b;
```

```
    1*   if x < y then
    2      a := x;
    3      b := y+a
         else
    4      a := y;
    5      b := x+a
         fi;
    6 !  a+b+x+y
end
```

```
> quit    # `выходим из режима отладчика`
Warning, computation interrupted
```

Чтение кодов библиотечных процедур

Иногда бывает необходимо прочитать код встроенной процедуры *Maple*, чтобы, например, понять, почему она дает не тот результат, который вы ожидаете.

С этой целью используется функция **interface** взаимодействия программы *Maple* с пользовательским интерфейсом. Эта функция используется для установки и запроса всех переменных, которые определяют формат вывода на дисплей, но не связаны с вычислениями.

Одна из переменных этой функции **verboseproc** определяет форму вывода на дисплей встроенных процедур *Maple*. По умолчанию эта переменная равна 1, при этом команда

```
> eval(имя процедуры);
```

выводит на дисплей пользовательские процедуры полностью, однако библиотечные процедуры только схематично в форме

proc(x) ... end;

Если ввести команду

```
> interface(verboseproc = 2);
```

устанавливающую для переменной **verboseproc** значение 2, то командой **eval** можно распечатать полный код библиотечных процедур, но не процедур ядра (которые, как уже упоминалось, написаны на языке Си). Примеры

```
> eval(finance[annuity]);
```

proc(*Cash, Rate, Nperiods*)

description '*Present value of an annuity paying Cash per period for Nperiods periods*'

...

end

8.1.4. Сохранение процедур и чтение их в сеансе *Maple*

Для сохранения кода созданной вами процедуры (или нескольких процедур) используется команда

save filename
или
save name1, name2, ..., namek, filename.

Если команда записана в первой форме, то она сохранит все присвоенные имена в указанном файле, а если во второй, — только перечисленные имена. Например, для сохранения всех созданных вами процедур в файле aff.m в каталоге e:*Maple*v4\Mylib\ наберите просто в командной строке

```
> save`e:/MapleV4/MyLib/aff.m`;
```

Теперь можно из любого сеанса *Maple* получить доступ к процедурам, записанным в файле aff.m, введя команду

```
> read `e:/MapleV4/MyLib/aff.m`;
```

8.1.5. Создание собственной библиотеки и оформление справки по ее командам

Теперь покажем, как можно пополнить библиотеку *Maple* созданной вами процедурой и ввести справку о ней в базу данных, чтобы можно было в дальнейшем пользоваться ею как собственной *Maple*-процедурой.

Предположим, что вы хотите использовать в дальнейшем созданную вами процедуру оператора аффинного преобразования для функции от одной переменной.

```
> restart;aff:=proc(f,a::constant,b::constant)
  unapply(f(a*x+b),x);
  end;
```

$$aff := \mathbf{proc}(f,\ a::constant,\ b::constant)\ \mathrm{unapply}(f(a*x+b),\ x)\ \mathbf{end}$$

```
> aff((x->x)*cos,Pi,exp(1))(x);
```

$$(\pi\, x + \mathbf{e})\cos(\pi\, x + \mathbf{e})$$

В том же пакете вы хотите сохранить команду построения графика, содержащего кривые исходной функции f и функции aff(f).

Назовем эту команду affplot

```
> affplot:=proc(f,a::constant,b::constant,r::range)
  plot([f,aff(f,a,b)],r);
  end;
```

$$affplot := \mathbf{proc}(f,\ a::constant,\ b::constant,\ r::range)\ \mathbf{plot}([f,\ \mathrm{aff}(f,\ a,\ b)],\ r)\ \mathbf{end}$$

Например, применяя ее к функции x*cos(x), получим (рис. 72)

```
> affplot((x->x)*cos,Pi,exp(1),0..Pi);
```

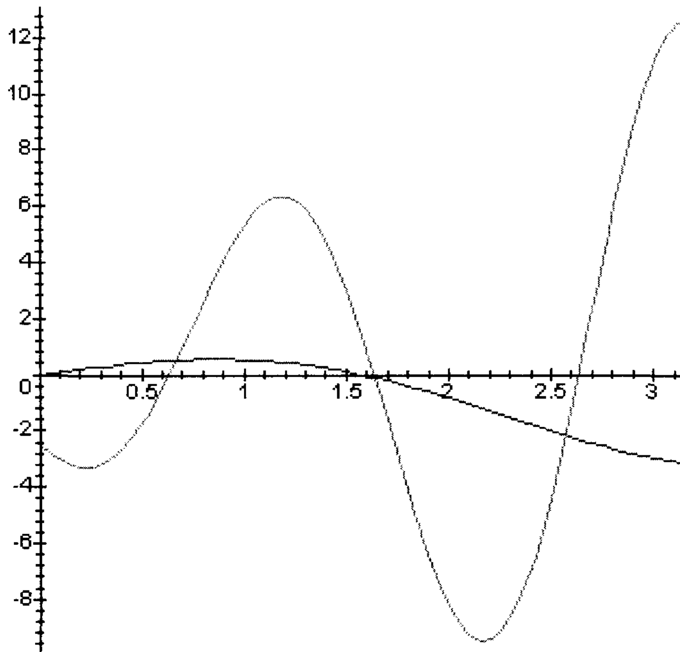

Рис. 72

Прежде всего нужно создать каталог вашей библиотеки. Пусть этот каталог **e:\MapleV4\Mylib**.

Теперь нужно записать в нее коды созданных вами команд

```
> save aff, affplot,`e:/MapleV4/MyLib/aff.m`;
```

Чтобы из любого сеанса *Maple* получить доступ к процедурам, записанным в файле aff.m, достаточно ввести команду:

```
> read `e:/MapleV4/MyLib/aff.m`;
```

Для создания справки по команде нужно в отдельном рабочем документе оформить текст справки с возможными примерами использования так, как вы хотели бы, чтобы выглядел ваш справочный файл. Если придерживаться стандартов справок *Maple*, то этот файл будет выглядеть следующим образом:

Оператор: aff — оператор аффинного преобразования функции

Функция: affplot — функция построения графика кривых исходной и преобразованной функций

Способ вызова:

aff(f, a, b)
afftplot(f, a, b, r)

Параметры:

f — функция от одной переменной
a — число
b — число
r — диапазон изменения переменной, имеет тип range и записывается в виде r1..r2, где r1, r2 — границы диапазона.

Описание:

- aff возвращает функцию, эквивалентную subs(x=a*x+b,f(x)).
- affplot возвращает график с изображением исходной функции f и функции aff(f).

Примеры:

```
> read `e:/MapleV4/MyLib/aff.m`:
  aff(sin*cos,a,b)(x);
```

Пример построения графика (рис. 73)

```
> affplot(sin*cos,exp(1),Pi/4,0..Pi);
```

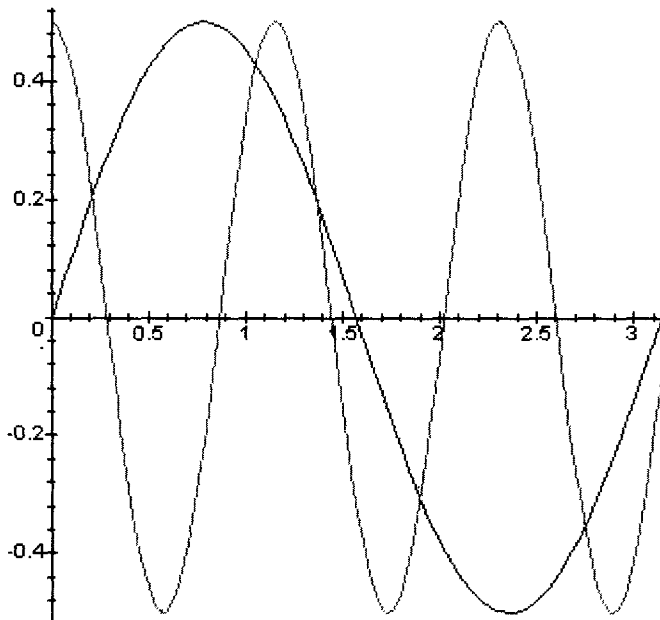

Рис. 73

Теперь нужно сохранить его в виде рабочего документа *Maple* (с расширением .mws) либо текстового файла (с расширением .txt). После этого вы должны включить его в базу данных справки. Для этого существует специальная команда **makehelp**. Она записывается в следующем виде:

makehelp(название темы, файл справки, библиотека), где

- название темы — название темы создаваемого файла справки;
- текстовый файл — имя текстового файла, содержащего текст справки;
- библиотека — библиотека, в которой должен быть сохранен файл справки.

Название темы должно быть записано в форме \`имя\`, или \`имя1/имя2\`. Если задан третий аргумент, **makehelp** должен сохранить файл в базе данных указанной библиотеки.

Если эта база данных будет находиться в созданном вами каталоге mylib, то ее имя будет \`e:\\\\Maple4\\\\mylib\\\\Maple.hdb\`. Предположим, что созданный вами файл справки сохранен под именем \`e:\\\\Maple4\\\\mylib\\\\aff.mws\`, тогда нужно ввести команды:

```
> readlib(makehelp):
> makehelp(`aff`,`e:/Maplev4/mylib/aff.mws`,
  `e:/Maplev4/mylib`):
> makehelp(`affplot`,`e:/Maplev4/mylib/aff.mws`,
  `e:/Maplev4/mylib`):
```

Справка помощи по пакету aff готова. Вы сможете получить доступ к справке помощи после того, как вы зарегистрируете каталог вашей библиотеки.

Вообще говоря, местонахождение (каталог) системных файлов основной библиотеки (или библиотек) определяется значением имени **libname**. Это обычно одно имя, но можно задать последовательность имен, определяющих последовательность каталогов, которые будут просматриваться по порядку при поиске команд.

Команда, определяющая нахождение дополнительной библиотеки mylib, вводится следующим образом:

```
libname := libname, `/mylib`;
```

После этого команды, такие как **readlib**() и **with**() просматривают обе библиотеки. Чтобы узнать текущее имя библиотеки достаточно ввести команду **libname**; в сессии *Maple*.

Можно также определить начальное имя библиотеки при загрузке программы, введя в строку загрузки опцию

```
Maple.exe -b `/mylib`,
```

определяющую нахождение дополнительной библиотеки.

В нашем случае для регистрации библиотеки пользователя нужно ввести команду:

```
> libname:=libname, `e:\\Maplev4\\mylib`;
```

libname := *E*:*MAPLEV4*/*update*, *E*:*MAPLEV4*/*lib*, *e*:*Maplev4**mylib*, *e*:*Maplev4**mylib*

Теперь введя из командной строки

```
> ?`affplot`
```

вы вызовите окно справки помощи по команде affplot.

8.1.6. Чтение и запись данных в файлы

Запись данных в файл

Вначале данные необходимо организовать в виде массива (списка, списка списков, матрицы)

```
> L := [[1., 19.53, 73., 51.96], [2., 18.7, 61.,
  44.84], [3., 21.69, 60., 46.11], [4., 25.37, 86.,
  57.62], [5., 24.43, 87., 55.17], [6., 19.09, 86.,
  46.57], [7., 18.3, 80., 48.03], [8., 24.72, 77.,
  54.62], [9., 21.88, 77., 51.17], [10., 18.8, 81.,
  50.54], [11., 17.79, 54., 39.81], [12., 5.13,
  34., 21.11], [13., 3.14, 29., 16.29], [14., -.33,
  20., 11.93], [15., 3.8, 33., 21.6], [16., 4.42,
  29., 20.68], [17., 5.72, 50., 31.12], [18., 7.99,
  42., 31.06], [19., 8.58, 27., 27.76], [20., 7.47,
  50., 33.81], [21., 8.75, 74., 47.03], [22., 5.45,
  67., 40.51], [23., 9.88, 80., 50.03], [24., 9.9,
  80., 48.84], [25., 9.67, 74., 45.07], [26., 1.67,
  50., 24.95], [27., 4.17, 48., 25.36], [28., 4.39,
  55., 27.63], [29., 2.85, 50., 25.4], [30., 2.67,
  67., 29.83], [31., 8.22, 89., 43.28], [32., 5.56,
  68., 33.45], [33., 9.03, 89., 43.61], [34., 7.68,
  83., 39.86], [35., 6.04, 77., 36.07], [36., 2.,
  52., 24.01], [37., .52, 32., 15.16], [38., -.4,
  24., 11.42], [39., -.74, 13., 6.1], [40., -.51,
  20., 9.48], [41., 0, 87., 5.26]]:
```

Затем при помощи команды **writedata** данные запоминаются в файл

```
> writedata(`e:\\Maplev4\\data2.txt`,L);
```

Чтение данных из файла

производится при помощи команды

```
> L:=readdata(`e:\\Maplev4\\data2.txt`,4);
```

L := [[1., 19.53, 73., 51.96], [2., 18.7, 61., 44.84], [3., 21.69, 60., 46.11],
[4., 25.37, 86., 57.62], [5., 24.43, 87., 55.17], [6., 19.09, 86., 46.57],
[7., 18.3, 80., 48.03], [8., 24.72, 77., 54.62], [9., 21.88, 77., 51.17],
[10., 18.8, 81., 50.54], [11., 17.79, 54., 39.81], [12., 5.13, 34., 21.11],
[13., 3.14, 29., 16.29], [14., −.33, 20., 11.93], [15., 3.8, 33., 21.6],
[16., 4.42, 29., 20.68], [17., 5.72, 50., 31.12], [18., 7.99, 42., 31.06],
[19., 8.58, 27., 27.76], [20., 7.47, 50., 33.81], [21., 8.75, 74., 47.03],
[22., 5.45, 67., 40.51], [23., 9.88, 80., 50.03], [24., 9.9, 80., 48.84],
[25., 9.67, 74., 45.07], [26., 1.67, 50., 24.95], [27., 4.17, 48., 25.36],
[28., 4.39, 55., 27.63], [29., 2.85, 50., 25.4], [30., 2.67, 67., 29.83],
[31., 8.22, 89., 43.28], [32., 5.56, 68., 33.45], [33., 9.03, 89., 43.61],
[34., 7.68, 83., 39.86], [35., 6.04, 77., 36.07], [36., 2., 52., 24.01],
[37., .52, 32., 15.16], [38., −.4, 24., 11.42], [39., −.74, 13., 6.1],
[40., −.51, 20., 9.48], [41., 0, 87., 5.26]]

где вторым параметром аргумента команды является число столбцов.

Данные можно соответствующим образом преобразовать

```
> g1:=map(u->[u[1],u[2]],L);
  g2:=map(u->[u[1],u[4]],L);
```

g1 := [[1., 19.53], [2., 18.7], [3., 21.69], [4., 25.37], [5., 24.43], [6., 19.09],
[7., 18.3], [8., 24.72], [9., 21.88], [10., 18.8], [11., 17.79], [12., 5.13],
[13., 3.14], [14., −.33], [15., 3.8], [16., 4.42], [17., 5.72], [18., 7.99],
[19., 8.58], [20., 7.47], [21., 8.75], [22., 5.45], [23., 9.88], [24., 9.9],
[25., 9.67], [26., 1.67], [27., 4.17], [28., 4.39], [29., 2.85], [30., 2.67],
[31., 8.22], [32., 5.56], [33., 9.03], [34., 7.68], [35., 6.04], [36., 2.],
[37., .52], [38., −.4], [39., −.74], [40., −.51], [41., 0]]

g2 := [[1., 51.96], [2., 44.84], [3., 46.11], [4., 57.62], [5., 55.17], [6., 46.57],
[7., 48.03], [8., 54.62], [9., 51.17], [10., 50.54], [11., 39.81], [12., 21.11],
[13., 16.29], [14., 11.93], [15., 21.6], [16., 20.68], [17., 31.12], [18., 31.06],
[19., 27.76], [20., 33.81], [21., 47.03], [22., 40.51], [23., 50.03], [24.,
48.84], [25., 45.07], [26., 24.95], [27., 25.36], [28., 27.63], [29., 25.4], [30.,
29.83], [31., 43.28], [32., 33.45], [33., 43.61], [34., 39.86], [35., 36.07],
[36., 24.01], [37., 15.16], [38., 11.42], [39., 6.1], [40., 9.48], [41., 5.26]]

и использовать, например, для построения графика (рис. 74)

```
> plot({g1,g2});
```

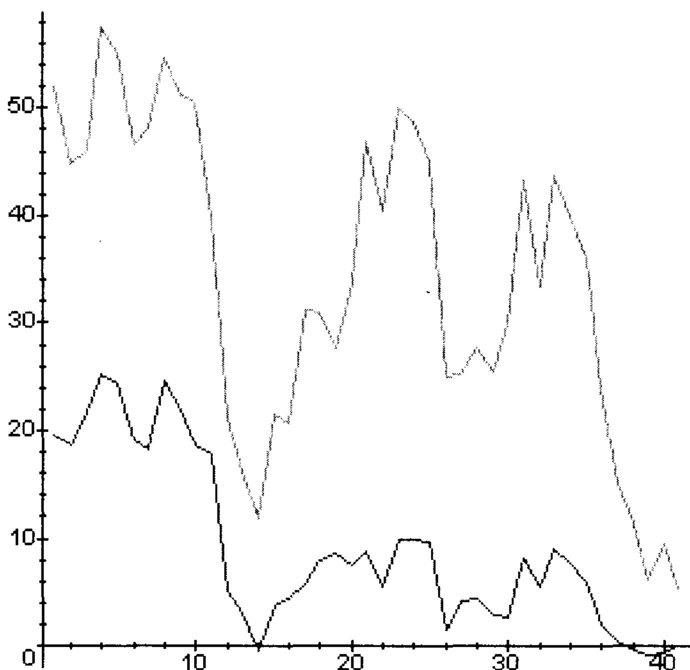

Рис. 74

При помощи статистического пакета данные можно проанализировать

```
> with(stats);
```

[*anova*, *describe*, *fit*, *importdata*, *random*, *statevalf*, *statplots*, *transform*]

```
> with(describe);
```

[*coefficientofvariation, count, countmissing, covariance, decile, geometricmean, harmonicmean, kurtosis, linearcorrelation, mean, meandeviation, median, mode, moment, percentile, quadraticmean, quantile, quartile, range, skewness, standarddeviation, sumdata, variance*]

Выберем, например, данные из четвертого столбца

```
> L[1..nops(L),4];
```

[51.96, 44.84, 46.11, 57.62, 55.17, 46.57, 48.03, 54.62, 51.17, 50.54, 39.81, 21.11, 16.29, 11.93, 21.6, 20.68, 31.12, 31.06, 27.76, 33.81, 47.03, 40.51, 50.03, 48.84, 45.07, 24.95, 25.36, 27.63, 25.4, 29.83, 43.28, 33.45, 43.61, 39.86, 36.07, 24.01, 15.16, 11.42, 6.1, 9.48, 5.26]

```
> mean("); # среднее значение
```

34.00365853

```
> g1:=map(u->u[2],L);g2:=map(u->u[4],L);
  # Выбираем данные из 1 и 4 столбцов
```

g1 := [19.53, 18.7, 21.69, 25.37, 24.43, 19.09, 18.3, 24.72, 21.88, 18.8, 17.79, 5.13, 3.14, −.33, 3.8, 4.42, 5.72, 7.99, 8.58, 7.47, 8.75, 5.45, 9.88, 9.9, 9.67, 1.67, 4.17, 4.39, 2.85, 2.67, 8.22, 5.56, 9.03, 7.68, 6.04, 2., .52, −.4, −.74, −.51, 0]

g2 := [51.96, 44.84, 46.11, 57.62, 55.17, 46.57, 48.03, 54.62, 51.17, 50.54, 39.81, 21.11, 16.29, 11.93, 21.6, 20.68, 31.12, 31.06, 27.76, 33.81, 47.03, 40.51, 50.03, 48.84, 45.07, 24.95, 25.36, 27.63, 25.4, 29.83, 43.28, 33.45, 43.61, 39.86, 36.07, 24.01, 15.16, 11.42, 6.1, 9.48, 5.26]

Построим график рассеяния (рис. 75)

```
> statplots[scatter2d](g1,g2);
```

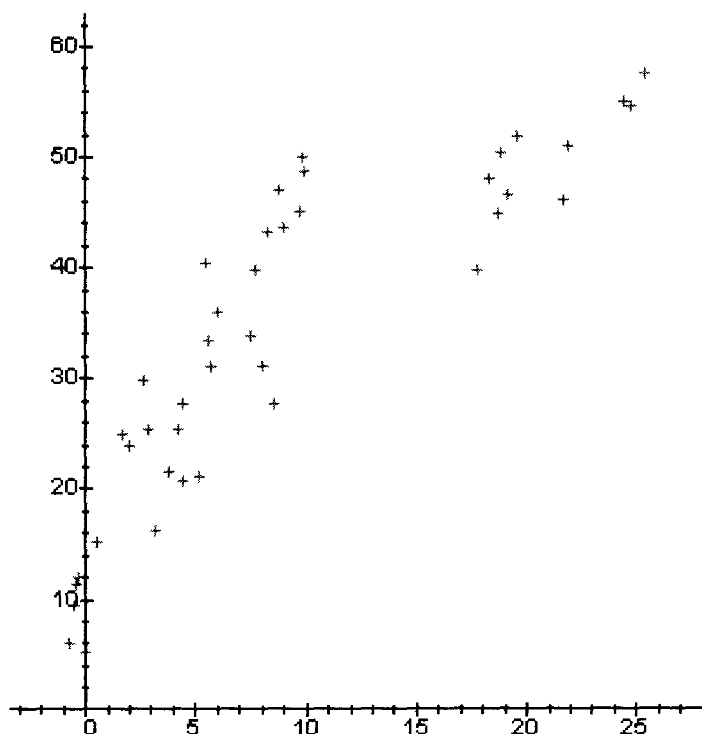

Рис. 75

Преобразуем формат данных в массив (матрицу)

```
> A:=convert(L,array);
```

A :=

1.	19.53	73.	51.96
2.	18.7	61.	44.84
3.	21.69	60.	46.11
4.	25.37	86.	57.62
5.	24.43	87.	55.17
6.	19.09	86.	46.57
7.	18.3	80.	48.03
8.	24.72	77.	54.62
9.	21.88	77.	51.17
10.	18.8	81.	50.54
11.	17.79	54.	39.81
12.	5.13	34.	21.11
13.	3.14	29.	16.29
14.	−.33	20.	11.93
15.	3.8	33.	21.6
16.	4.42	29.	20.68
17.	5.72	50.	31.12
18.	7.99	42.	31.06
19.	8.58	27.	27.76
20.	7.47	50.	33.81
21.	8.75	74.	47.03
22.	5.45	67.	40.51
23.	9.88	80.	50.03
24.	9.9	80.	48.84
25.	9.67	74.	45.07
26.	1.67	50.	24.95
27.	4.17	48.	25.36
28.	4.39	55.	27.63
29.	2.85	50.	25.4
30.	2.67	67.	29.83
31.	8.22	89.	43.28
32.	5.56	68.	33.45
33.	9.03	89.	43.61
34.	7.68	83.	39.86
35.	6.04	77.	36.07
36.	2.	52.	24.01
37.	.52	32.	15.16
38.	−.4	24.	11.42
39.	−.74	13.	6.1
40.	−.51	20.	9.48
41.	0	87.	5.26

Найдем произведение обратной матрицы на матрицу A

```
> transpose(A)&*A;
```

(transpose(A)) &* A

Выведем на дисплей вид полученной матрицы

```
> evalm(");
```

$$\begin{bmatrix} 23821. & 4827.55 & 48551. & 25049.39 \\ 4827.55 & 5928.1460 & 26355.82 & 16773.2910 \\ 48551. & 26355.82 & 163351. & 92848.60 \\ 25049.39 & 16773.2910 & 92848.60 & 56223.1325 \end{bmatrix}$$

8.1.7. Перекодировка процедур на языки Си и Фортран

Для перекодировки процедур на язык Си используется команда **C**(выражение, опция), где "выражение" — любое выражение *Maple*, массив выражений или процедура.

Необязательная опция может быть одной из следующих:

- filename=name — полученный код выводится в файл с именем name;
- optimized — выполняется оптимизация Си-кода;
- digits=n — все константы преобразуются в числа с плавающей точкой с n знаками точности;
- precision =single или precision=double — двойная или ординарная точность констант. Команда **C** должна вызываться с помощью строки readlib(C).

Примеры:

- Многочлен

```
> readlib(C):
  f := 1-2*x+3*x^2-2*x^3+x^4;
```

$$f := 1 - 2\,x + 3\,x^2 - 2\,x^3 + x^4$$

```
> C(f);
```

```
      t0 = 1.0-2.0*x+3.0*x*x-2.0*x*x*x+pow(x,4.0);
```

```
> C(f,optimized);
```

```
      t1 = x*x;
      t3 = t1*t1;
      t4 = 1.0-2.0*x+3.0*t1-2.0*t1*x+t3;
```

♦ Выражение с элементарными функциями и константами

```
> f := Pi*ln(x^2)-sqrt(2)*ln(x^2)^2;
```

$$f := \pi \ln(x^2) - \sqrt{2} \ln(x^2)^2$$

```
> C(f);
```

```
t0 = 0.3141592653589793E1*log(x*x)-sqrt(2.0)*pow(log(x*x),
2.0);
```

```
> C(f,optimized);
```

```
t1 = x*x;
t2 = log(t1);
t4 = sqrt(2.0);
t5 = t2*t2;
t7 = 0.3141592653589793E1*t2-t4*t5;
```

♦ Массив выражений

```
> v := array([exp(-x)*x,exp(-x)*x^2,exp(-x)*x^3]);
```

$$v := [e^{(-x)}\, x,\ e^{(-x)}\, x^2,\ e^{(-x)}\, x^3]$$

```
> C(v,optimized);
```

```
t1 = exp(-x);
t3 = x*x;
v[0] = t1*x;
v[1] = t1*t3;
v[2] = t1*t3*x;
```

♦ Матрица с неопределенным элементом

```
> A := array(1..2,1..2,symmetric):
A[1,1] := log(x): A[1,2] := 1-log(x):
print(A);
```

$$\begin{bmatrix} \ln(x) & 1 - \ln(x) \\ 1 - \ln(x) & A[2, 2] \end{bmatrix}$$

```
> C(A,precision=double);
```

```
A[0][0] = log(x);
A[0][1] = 1.0-log(x);
A[1][0] = 1.0-log(x);
A[1][1] = undefined;
```

```
> C(A,optimized);
      t1 = log(x);
      t2 = 1.0-t1;
      A[0][0] = t1;
      A[0][1] = t2;
      A[1][0] = t2;
      A[1][1] = undefined;
```

Для перекодировки процедур *Maple* на язык *Фортран* используется команда **fortran**, вызываемая и действующая аналогично команде **С**.

8.2. Программирование свойств и правил вычисления функций и операторов

Предположим, что мы хотим ввести в *Maple V* совершенно новую функцию, которую невозможно выразить через другие элементарные или специальные функции, имеющиеся в программе. Какие программные средства имеются в *Maple* для задания свойств этой функции, правил преобразования и упрощения выражений, содержащих эту функцию, а также получения численных значений и построения графика?

8.2.1. Команда *define*

Команда **define** задает свойства оператора или функции, ее можно применять тремя различными способами.

Способ 1

Можно определить оператор, как принадлежащий некоторому абстрактному алгебраическому пространству аа командой **define**(аа(оператор)) (в настоящее время программа позволяет определять пространство аа как линейное (аа=**Linear**) или как группу (аа=**Group**)).

Например, введем оператор h как линейный

```
> define(Linear(h));
```

Если мы воздействуем этим оператором на некоторую функцию, скажем многочлен 2*x+3*x^2, то получим

```
> h(2*x+3*x^2);
```

$$2\ \mathrm{h}(x) + 3\ \mathrm{h}(x^2)$$

То есть мы видим, что выражения, содержащие этот оператор, упрощаются. Это не случайно, при задании оператора командой **define** автоматически создаются входы для таких команд, как **simplify, eval, expand, diff, series**, и **testeq** так, чтобы осуществить необходимые свойства оператора.

Если абстрактное алгебраическое поле — группа, то команда **define** записывается следующим образом:

define(**Group**(Имя оператора, Единица, Оператор обращения)), где

- "Имя оператора" — имя бинарного оператора, определяемого на группе;
- "Единица" — символ единицы (единичного элемента) группы;
- "Оператор обращения" — унарный оператор, который вычисляет обратный элемент для любого элемента группы.

При такой записи команда **define** задает оператор над группой, тем самым, задавая правила вычислений, преобразований и упрощений этого оператора.

Свойства группы:

- замкнутость;
- ассоциативность;
- единичный оператор действует как слева так и справа;
- оператор обращения — унарный оператор.

Приведем пример

```
> define(Group(`&+`, E, `&-`));
> z &+ E;
```

$$z$$

```
> &- (&- a);
```

$$a$$

```
> a &+ (&- a);
```

$$E$$

Способ 2

Можно определить оператор заданием некоторых свойств. В этом случае команду define нужно применять в следующей записи:

define(оператор, свойство.1, свойство.2, ...), где "свойство.i" — наименование свойства оператора.

Операторы могут использоваться в инфиксной нотации (&-имена):

```
a &+ b;          a &* b &* c;          &A x
```

или в функциональной нотации (любое имя):

```
Op(a,b);          Op(a);          Op(a,b,c)
```

В настоящее время *Maple* позволяет задавать следующие свойства:

- **unary** — свойство унарности оператора;
- **binary** — свойство бинарности;
- **associative** — свойство ассоциативности: f(x,f(y,z)) = f(f(x,y),z) = f(x,y,z);
- **commutative** (или **symmetric**) — свойство коммутативности (симметричности): f(x,y) = f(y,x);

- **antisymmetric** — свойство антисимметричности: f(x,y) = –f(y,x);
- **inverse**=g — определяет обратный (для унарного) оператор к оператору g;
- **identity**=x — выражение x, примененное к оператору f как слева так и справа дает единичный оператор;
- **zero**=x — определяет x как нулевой элемент оператора f, так что если любой аргумент f есть x то результат также x.

Приведем пример

```
> define(`&A`, associative, commutative, identity=0,
  zero=y);
> x &A (t &A z) &A (0 &A x);t &A z;0 &A x;
  x &A (t &A z);
```

$$\&A(t, z, x, x)$$

$$t \mathbin{\&A} z$$

$$x$$

$$\&A(t, z, x)$$

```
> define(f, commutative, associative, inverse=g);
Warning: new definition for   f
> f( g(x), x,y);
```

y

```
> f( g(x),f(x));
```

f()

```
> f( g(x), z, f(x,y), y );
```

f(y, y, z)

Способ 3

Можно задать конкретные правила выполнения оператора при помощи уравнений. В этом случае команда **define** записывается следующим образом

define(F,**forall**(*переменные*, F(*аргумент.1*)=*результат.1*, F(*аргумент.2*)=*результат.2*, ...))

где

- переменные — имя или имя с присвоенным типом, или список имен;
- F — имя определяемого при помощи команды **define** оператора;

- аргумент.i — аргументы, с которыми данный оператор будет использоваться;
- результат.i — программированные результаты применения F(аргумент.i);
- выражение **forall**() — необходимый параметр команды **define**, который обеспечивает назначение свойств оператору.

Соотношение F(аргумент.i)=результат.i вынуждает оператор выдавать результат.i каждый раз, когда он получает аргумент.i. Заданные в команде "переменные" могут быть именем или списком имен, они используются для сравнения, должны появляться строго один раз и могут иметь любое значение. Замена затем будет выполнена для всех таких значений. Если имена приведены в форме **type**(имя), где **type** является одним из имеющихся в *Maple* типов объектов и имя — некоторое имя, то имя будет сравниваться только в случае, если оно имеет указанный тип.

Приведем пример.

```
> restart;define(F,forall([n,x],F(x^n)=n*F(x)),
  F(sin(x))=exp(x+b)); # задаем при помощи команды
  define свойства оператора F

> F(sin(x)^3.5)+F(1/sin(x)); # результат действия
  оператора F
```

$$2.5\,e^{(x+b)}$$

8.2.2. Программирование правил вычисления

Если мы хотим, чтобы выражения, включающие заданную нами функцию, преобразовывались, упрощались и вычислялись по некоторым правилам, необходимо эти правила задать. Программа *Maple V* имеет программируемые входы для многих команд, таких как **simplify**, **diff**, **expand**, **series**, **evalf**, и многих других функций.

Например, чтобы задать правила дифференцирования некоторой функции H, нужно определить процедуру с именем `` `diff/H` ``, задающую правило дифференцирования H.

Если процедура `` `diff/H` `` определена, то вызов функции **diff**(H(x, y, ...), y) будет вызывать процедуру `` `diff/H` ``(x,y,...,y) для вычисления производной.

В качестве примера научим *Maple* вычислять производную функции H от одной переменной H(x)

```
> restart;`diff/H` := proc(x,y) H(x)*diff(x,y)/x end:
  # задаем правило вычисления производной функции H

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

$$\frac{H(x)}{x}$$

```
> diff(H(g(x)),x);
```

$$\frac{H(g(x))\left(\frac{d}{dx}g(x)\right)}{g(x)}$$

```
> diff(H(sin(x)),x);
```

$$\frac{H(\sin(x))\cos(x)}{\sin(x)}$$

```
> evalf(subs(x=Pi/2,"));
```

$$-.2051033807\ 10^{-9}\ H\left(\sin\left(\frac{1}{2}\pi\right)\right)$$

Из примеров видно, что наша процедура только дополняет известные и имеющиеся в *Maple* правила вычисления производных.

Точно также, пользователь может задать собственные правила упрощения при помощи некоторой процедуры. Если процедура `` `simplify/f` `` задана, то вызов функции **simplify**(a,f) задействует `` `simplify/f` ``(a).

Научим, например *Maple* упрощениям H(x^n)=n*H(x) и H(x*y)=H(x)+H(y).

```
> restart;
`simplify/H` := proc(y)
local x;
    if op(0, y) = H then
        if type(op(y), anything^numeric) then x :=
        op(1, op(y));
        elif type(op(y), `*`) then
  RETURN(H(op(1, op(y))) + H(op(2, op(y))));
        else RETURN(y)
        fi;
        op(2, op(y))*H(x);
    else y
    fi;
end: # задаем правила упрощения функции H
```

Теперь можно проверить как эти правила действуют

```
> symplify(H(sin(y)^5),H);
```

$$5\ \mathrm{H}(\sin(y))$$

```
> simplify(H(sin(x)*cos(y)),H);
```

$$\mathrm{H}(\sin(x)) + \mathrm{H}(\cos(y))$$

```
> expand(ln(sin(x)^n),H);
```

$$n\ \ln(\sin(x))$$

Функции **expand** и **series** также можно научить выполнять операции над введенными пользователем функциями. Если процедура `` `expand/f` `` задана, то выполнение **expand**(f(x)) будет приводить к выполнению `` `expand/f` ``(x);

Один из методов задания численных значений определенной пользователем функции f – при помощи задания правил разложения ее в степенной ряд. Пользователь может "научить" команду **series** разлагать функцию в ряд при помощи процедуры `` `series/f` ``. Если процедура `` `series/f` `` задана, то выполнение команды **series**(f(x,y),x) вызовет выполнение процедуры `` `series/f` ``(x,y,x) для вычисления ряда. Заметим, что процедура `` `series/f` `` будет создавать объект типа **series**, а не полином (**polynomial**). Чтобы получить вычисляемый полином, нужно применить команду **convert**(выражение, polynom).

8.2.3. Сравнение с шаблоном

Еще две очень полезные функции, помогающие программировать правила вычислений: **match** — сравнение с шаблоном и **typematch** — сравнение типов.

Функция **match** вызывается следующей командой **match**(выражение = шаблон, v, `` `s` ``), где

- выражение — сравниваемое выражение
- шаблон — шаблон, с которым сравнивается выражение
- v — имя **главной** переменной
- `` `s` `` — имя возвращаемого аргумента

Функция **match** возвращает **true**, если она может подтвердить соответствие шаблону и **false** в противном случае. Если сравнение удачно, то величине s присваивается система равенств, подстановка которых в шаблон приведет к сравниваемому выражению.

Главная переменная — именно та переменная, относительно которой происходит сравнение с шаблоном.

Сравнение командой **match** является математическим сравнением, иначе говоря, выражения вычисляются, если возможно, чтобы удовлетворить шаблону. Команда **typematch** сравнивает только форму объектов. Приведем пример

```
> match(exp(x)/x^(2) = A*exp(x)^P*x^Q, x, `s`);
```

true

```
> s;
```

$$\{A = 1,\ P = 1,\ Q = -2\}$$

Заметим, что функция **match** может быть очень полезна также, когда из сложного выражения необходимо выделить члены, соответствующие некоторому шаблону.

Функция **typematch** осуществляет сравнение выражений по типу переменных. Она вызывается одной из следующих команд:

typematch(выражение, t) или

typematch(выражение, t, 's'),

где

выражение — любое выражение

t — выражение, содержащее переменные с указанием их типов '::' — оператором

s — (необязательная опция) — имя

Команда **typematch** является логической функцией. Она возвращает **true**, если вводимое "выражение" удовлетворяет типу t. Обычно t содержит выражения вида v_i :: t_i где v_i переменная, t_i — ее тип.

Если сравнение командой **typematch**(e,t,'s') успешно, параметру s присваивается список уравнений вида v_i = r_i.

Заметим, что аргументы оператора связи '::' шаблона вычисляются. Таким образом, если переменной x может быть присвоено некоторое значение — например, в предыдущей команде сравнения — то, чтобы не было ошибки, ее необходимо заключать в кавычки. То есть записать команду в виде **typematch**(e,'x'::name=range).

Приведем примеры

```
> typematch( y^3, b::name^(n::integer), 's' );
```

$$true$$

```
> s;
```

$$[b = y,\ n = 3]$$

```
> typematch( [a,b,c,d,e], list( v::name ), 's' );
```

$$true$$

```
> s;
```

$$[v = a,\ v = b,\ v = c,\ v = d,\ v = e]$$

Команда **typematch** может быть также использована для разбиения сложного выражения на элементы, тип которых удовлетворяет шаблону сравнения.

8.3. Пакет *Domains*

Domains — новый пакет *Maple*. Его идея пришла из программы *AXIOM*, и состоит в том, чтобы передавать в качестве параметра в аргументах процедур набор функций как одно целое (называемое домен (**domain**)), например кольцо коэффициентов кольцу полиномов. При этом код программ вычисления полиномов не будет меняться при изменении домена кольца коэффициентов, то есть он более универсальный, чем код программ *Maple*. Этот пакет предполагается использовать как средство для разработки сложных алгоритмов. Предполагается заменить в следующих версиях *Maple* коды некоторых библиотек на более универсальные коды **Domains**.

8.3.1. Домены в *Domains*

Домены являются функциями, которые возвращают таблицы операций для вычислений в данном домене. Например, **Integer**() возвращает таблицу операций для вычислений с целыми числами, включающую `+` — сложение, `-` — вычитание, `*` — умножение и так далее.

Все домены принадлежат множеству категорий, которое поддерживает операции

```
=, <>  - логические отношения элементов;
Input  - для конвертирования выражений в представление дан-
ных Domains;
Output - для обратного конвертирования;
Random - для генерации псевдослучайных величин области;
Type   - для проверки, является ли величина элементом области.
```

Список доменов, сконструированных в настоящее время в **Domains** следующий:

```
Z     Integer();
Q     Rational();
G     Gaussian(R:Ring);
Zmod  Zmod(n:posint);
GF    GaloisField(p:prime, k:posint);

DUP   DenseUnivariatePolynomial(R:Ring, x:name);
OUP   OrderedUnivariatePolynomial(P:UnivariatePolynomial(R),
      f:(R,R) -> Boolean);

DEV   DenseExponentVector(X:list(name));
PEV   PrimeExponentVector(X:list(name));
MEV   MapleExponentVector(X:list(name));
TEV   MacaulayExponentVector(X:list(name));
TDMP  TableDistributedMultivariatePolynomial(R:Ring, E:Ex-
      ponentVector);
SDMP  SparseDistributedMultivariatePolynomial(R:Ring,
      E:ExponentVector);
```

```
QF    ExpandedNormalFormQuotientField(D:GcdDomain);
ENFQF ExpandedNormalFormQuotientField(D:GcdDomain);
FNFQF FactoredNormalFormQuotientField(D:GcdDomain);
RF    RationalFunction(D:GcdDomain, X:list(name));
LUPS  LazyUnivariatePowerSeries(R:Ring, x:name);
      Matrix(R:Ring);

SM    SquareMatrix(n:posint, R:Ring);
SAE   AlgebraicExtension(D:UnvivaraitePolynomial, m:D).
```

Кроме того, имеется несколько специальных доменов, использующих *Maple*-представление полиномов с целью получения большей эффективности для целых и рациональных коэффициентов.

```
MUP MapleUnivariatePolynomial(R:Z, Q, Zmod, x:name);
MMP MapleMultivariatePolynomial(R:Z, Q, Zmod, X:list(name)).
```

Вывести все операции, доступные для данного домена можно при помощи команды **show**(D, operations).

8.3.2. Примеры использования пакета *Domains*

На нескольких примерах будет показано как использовать **Domains**. Заметим, что все функции **Domains** начинаются с заглавной буквы. Вначале загрузим пакет **Domains**.

```
> with(Domains);
```

```
--------------------Domains version 1.0 --------------------
Initially defined domains are Z and Q the integers and rationals
Abbreviations, e.g. DUP for DenseUnivariatePolynomial, also made
```

[*init*]

При загрузке были определены области **Z** (целых чисел) и **Q** (рациональных чисел). Давайте проделаем несколько вычислений. Найдем наибольший общий делитель чисел 345 и 60:

```
> Z[Gcd](345,60);
```

15

Перемножим несколько рациональных чисел:

```
> Q[`*`](3/2,7/3,12/5);
```

42/5

Проверим, что Z и Q являются таблицами *Maple* :

```
> type(Z,table);
```

true

Они содержат операции (*Maple* процедуры) для вычислений в **Z** и **Q.** Какие операции доступны в Q?

```
> show(Q,operations);
```

```
Signatures for constructor Q
note: operations prefixed by  -  are not available

* : (Q,Q*) -> Q
* : (Integer,Q) -> Q
+ : (Q,Q*) -> Q
- : Q -> Q
- : (Q,Q) -> Q
/ : (Q,Q) -> Q
/ : (Q,Integer) -> Q
0 : Q
1 : Q
< : (Q,Q) -> Boolean
<= : (Q,Q) -> Boolean
<> : (Q,Q) -> Boolean
= : (Q,Q) -> Boolean
> : (Q,Q) -> Boolean
>= : (Q,Q) -> Boolean
AbsoluteDegree : Integer
BaseDomain : IntegralDomain
Characteristic : Integer
Coerce : Integer -> Q
Denom : Q -> G
Div : (Q,Q) -> Union(Q,FAIL)
EuclideanNorm : Q -> Integer
Factor : Q -> [Q,[[Q,Integer]*]]
Gcd : Q* -> Q
Gcdex : (Q,Q,Name,Name) -> Q
Gcdex : (Q,Q,Name) -> Q
Input : Expression -> Union(Q,FAIL)
Inv : Q -> Union(Q,FAIL)
Lcm : Q* -> Q
Max : (Q,Q*) -> Q
Min : (Q,Q*) -> Q
ModularHomomorphism : () -> (Q -> Union(Integer,FAIL),
Integer)
```

```
Normal : Q -> Q
Numer : Q -> G
Output : Q -> Expression
Powmod : (Q,Integer,Q) -> Q
Prime : Q -> Boolean
Quo : (Q,Q) -> Q
Quo : (Q,Q,Name) -> Q
Random : () -> Q
RelativelyPrime : (Q,Q) -> Boolean
Rem : (Q,Q,Name) -> Q
Rem : (Q,Q) -> Q
Sign : Q -> UNION(1,-1,0)
Slash : (G,G) -> Q
SmallerEuclideanNorm : (Q,Q) -> Boolean
Sqrfree : Q -> [Q,[[Q,Integer]*]]
Type : Expression -> Boolean
Unit : Q -> Q
UnitNormal : Q -> [Q,Q,Q]
Zero : Q -> Boolean
^ : (Q,Integer) -> Q
```

Далее выполним некоторые операции в области полиномов от одной переменной над Q. Вначале нужно создать домен в Q[x], назовем его C.

```
> C := DenseUnivariatePolynomial(Q,x):
```

Имя **DenseUnivaritePolynomial** указывает, что используемая структура данных является плотной. Каждый домен пакета **Domains** имеет функцию **Random**. Эта функция возвращает случайно сгенерированный элемент из домена, который можно использовать для написания примеров.

Введем случайный полином.

```
> a := C[Random]();
```

$$a := -\frac{35}{97} + \frac{50}{79}x + \frac{55}{37}x^2$$

Теперь выведем полином m

```
> C[Output](a);
```

$$-\frac{35}{97} + \frac{50}{79}x + \frac{55}{37}x^2$$

Можно вычислить степень полинома.

```
> C[Degree](a);
```

$$2$$

Квадрат полинома

```
> C['^'](a,2);
```

$$\frac{1225}{9409}-\frac{3500}{7663}x-\frac{15055350}{22398949}x^2+\frac{5500}{2923}x^3+\frac{3025}{1369}x^4$$

Domains может также оперировать с матрицами и другими объектами линейной алгебры. Вычислим, например, обратную матрицу к матрице Гильберта 3×3. Вначале мы должны создать матричный домен:

```
> M3 := SquareMatrix(3,Q):
> A := M3[Input]([[1,2,3],[1,1/2,1/3],[3,1/4,5]]);
```

$$A := \left[[1, 2, 3], \left[1, \frac{1}{2}, \frac{1}{3}\right], \left[3, \frac{1}{4}, 5\right]\right]$$

```
> M3[Det](A);
```

$$\frac{-28}{3}$$

```
> M3[Inv](A);
```

$$\left[\left[\frac{-29}{112}, \frac{111}{112}, \frac{5}{56}\right], \left[\frac{3}{7}, \frac{3}{7}, \frac{-2}{7}\right], \left[\frac{15}{112}, \frac{-69}{112}, \frac{9}{56}\right]\right]$$

Давайте теперь вычислять матрицы полиномов в Q[x]. Будем использовать матрицы 2×2,чтобы примеры не получились очень громоздкими.

```
> M2 := SquareMatrix(2,C):
> A := M2[Input]([[1,x^3],[1-x,1+x^3]]);
```

$$A := [[1, x^3], [1-x, 1+x^3]]$$

Вычислим определитель матрицы

```
> C[Output]( M2[Det](A) );
```

$$x^4+1$$

Вычислим квадрат матрицы

```
> M2[Output](M2['^'](A,2));
```

$$3[[-x^4+x^4+1, x^6+2x^3], [-x^4+x^3+2-2x, x^6+1-x^4+3x^3]]$$

Вычислим обратную матрицу

```
> M2[Output]( M2[Inv](A));
```

```
Error, (in notImplemented) operation is not implemented
```

Программа вывела сообщение об ошибке: нельзя вычислить обратную матрицу от матрицы полиномов, так как результат в общем случае — рациональная функция (не полином).

В следующих примерах приводятся вычисления со степенными рядами от одной переменной. Вначале создадим домен обрезанных степенных рядов от x над Q.

```
> PS := LazyUnivariatePowerSeries(Q,x):>
  show(PS, operations);
```

```
Signatures for constructor PS
note: operations prefixed by  –  are not available

* : (PS,PS*) -> PS
* : (Integer,PS) -> PS
+ : (PS,PS*) -> PS
- : (PS,PS) -> PS
- : PS -> PS
/ : (PS,Integer) -> PS
/ : (PS,PS) -> PS
0 : PS
1 : PS
<> : (PS,PS) -> Boolean
= : (PS,PS) -> Boolean
AbsoluteDegree : Integer
Characteristic : Integer
Coeff : (PS,Integer) -> Q
CoefficientRing : Ring
Coerce : Integer -> PS
Constant : Q -> PS
Cos : PS -> Union(PS,FAIL)
Cosh : PS -> Union(PS,FAIL)
Diff : PS -> PS
Div : (PS,PS) -> Union(PS,FAIL)
EuclideanNorm : PS -> Integer
Exp : PS -> Union(PS,FAIL)
Factor : PS -> [PS,[[PS,Integer]*]]
Gcd : PS* -> PS
Gcdex : (PS,PS,Name) -> PS
Gcdex : (PS,PS,Name,Name) -> PS
Input : Expression -> Union(PS,FAIL)
Integrate : PS -> Union(PS,FAIL)
```

```
Inv : PS -> PS
Lcm : PS* -> PS
Ln : PS -> Union(PS,FAIL)
Log : PS -> Union(PS,FAIL)
Lorder : PS -> Integer
Monomial : () -> PS
Monomial : Integer -> PS
Normal : PS -> PS
Output : PS -> Expression
Powmod : (PS,Integer,PS) -> PS
Prime : PS -> Boolean
Quo : (PS,PS,Name) -> PS
Quo : (PS,PS) -> PS
R* : (Q,PS) -> PS
R/ : (PS,Q) -> PS
Random : () -> PS
RelativelyPrime : (PS,PS) -> Boolean
Rem : (PS,PS,Name) -> PS
Rem : (PS,PS) -> PS
Series : [[Q,Integer]] -> PS
Shift : (PS,Integer) -> PS
Sin : PS -> Union(PS,FAIL)
Sinh : PS -> Union(PS,FAIL)
SmallerEuclideanNorm : (PS,PS) -> Boolean
Sqrfree : PS -> [PS,[[PS,Integer]*]]
Tan : PS -> Union(PS,FAIL)
Tanh : PS -> Union(PS,FAIL)
Type : Expression -> Boolean
Unit : PS -> PS
UnitNormal : PS -> [PS,PS,PS]
Variable : Name
^ : (PS,Rational) -> PS
^ : (PS,PS) -> PS
^ : (PS,Integer) -> PS
order : PS -> Integer
```

Давайте вычислим степенной ряд функции sin(x)

```
> a := PS[Input](x);
```

$$a := x$$

```
> s := PS[Sin](a);
```

$$s := x - \frac{1}{6}x^3 + \frac{1}{120}x^5 + O(x^7)$$

```
> s1:=PS['Diff'](s,x); # sin(x)^(1/2)
```

$$s1 := 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 + O(x^6)$$

Обрезанные степенные ряды "ленивы" ("lazy"), это означает, что коэффициенты вычисляются не сразу, а по требованию, то есть вычисляются только то количество коэффициентов, которое указано в команде. Например, мы можем вычислить члены ряда до десятой степени.

```
> PS[Output](s1,10);
```

$$1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 - \frac{1}{720}x^6 + \frac{1}{40320}x^8 - \frac{1}{3628800}x^{10} + O(x^{11})$$

Далее вычислим полиномы Чебышева по их производящей функции:

(1−x*t)/(1−2*x*t+t^2) = sum(T(n,x)*t^k, k=0..).

Вначеле создадим домен LUPS == LazyUnivariatePowerSeries.

```
> P := LUPS(LUPS(Q,x),t):
```

Вводим выражение

```
> p1 := P[Input](1-2*x*t+t^2);
```

$$p1 := 1 - 2\,x\,t + t^2$$

```
> p2:=P[Input](1-t*x);
```

$$p2 := 1 - x\,t$$

Теперь находим разложение в ряд по t величины p^(−1/2)

```
> P['/'](p2, p1);
```

$$1 + x\,t + (-1 + 2\,x^2)\,t^2 + (-3\,x + 4\,x^3)\,t^3 + (1 - 8\,x^2 + 8\,x^4)\,t^4 +$$
$$+ (5\,x - 20\,x^3 + 16\,x^5)\,t^5 + O(t^6)$$

Сравним полученную формулу с формулами для полиномов Чебышева, имеющихся в *Maple*

```
> seq(orthopoly[T](i,x), i=0..5);
```

$$1,\ x,\ -1 + 2\,x^2,\ -3\,x + 4\,x^3,\ 1 - 8\,x^2 + 8\,x^4,\ 5\,x - 20\,x^3 + 16\,x^5$$

8.3.3. Пакет *Domains* в интерактивном режиме

Для удобства интерактивного использования команд пакета **Domains** используется команда **evaldomains**[D](expr). **evaldomains** обеспечивает *Maple*-подобный пользовательский интерфейс, сортировку данных и проверку типов аргументов **Domains**-функций. Она вычисляет выражение *Maple* в домене D и возвращает выражение *Maple*. Приведем примеры.

Используем введенный нами выше домен C полиномов от одной переменной. Введем два случайных полинома a1 и a2.

```
> a1:=C[Random]();a2:=C[Random]();
```

$$a1 := -\frac{57}{59} - \frac{45}{8}x - \frac{93}{92}x^2 - \frac{43}{62}x^3 + \frac{7}{9}x^4$$

$$a2 := -\frac{5}{99} + \frac{61}{50}x + \frac{11}{9}x^2$$

Так выглядит операция перемножения полиномов в **Domains** — синтаксисе

```
> C[Output](C[`*`](a1,a2));
```

$$\frac{77}{81}x^6 + \frac{706}{6975}x^5 - \frac{67369157}{31764150}x^4 - \frac{56986531}{7058700}x^3 - \frac{28632029}{3582480}x^2 - \frac{116113}{129800}x + \frac{95}{1947}$$

Использование знакомого *Maple* синтаксиса для той же операции при помощи **evaldomains**

```
> evaldomains[C](a1*a2);
```

$$\frac{77}{81}x^6 + \frac{706}{6975}x^5 - \frac{67369157}{31764150}x^4 - \frac{56986531}{7058700}x^3 - \frac{28632029}{3582480}x^2 - \frac{116113}{129800}x + \frac{95}{1947}$$

Если необходимо вычислить несколько выражений в одном и том же домене удобно создать домен-вычислитель с использованием команды создания псевдонимов **alias**:

```
alias (dup = evaldomains[домен]):
```

Так как выражение, направляемое на **evaldomains**, вначале обрабатывается синтаксическим анализатором *Maple*, пользователь не должен забывать о встроенных упрощениях, перегруппировке операндов и вычислении функций верхнего уровня этим анализатором. Например, следующее выражение всегда приводится к 0.

```
evaldomains[C] (C[Random]() - C[Random]());
```

$$0$$

Если домен имеет некоммутативное умножение или сложение, должна быть использована &-версия оператора с целью предотвращения перегруппировки операндов. Например, используя введенный нами выше домен М3 квадратных матриц 3*3, введем псевдоним

```
alias (m = evaldomains[M3]);
```

$$I, m$$

```
> A := m (Random());
```

$$A := \left[\left[\frac{2}{3}, \frac{-31}{26}, -62\right], \left[\frac{47}{91}, \frac{47}{61}, \frac{-41}{58}\right], \left[\frac{-90}{53}, \frac{-1}{94}, \frac{-83}{86}\right]\right]$$

```
> m (A &* (A + 1));
```

$$\left[\left[\frac{119379511}{1128582}, \frac{-251162}{111813}, \frac{-8297429}{194532}\right], \left[\frac{62943379}{25595661}, \frac{18140237257}{23999393236},\right.\right.$$

$$\left.\left.\frac{-902400427}{27688388}\right], \left[\frac{-496409}{414778}, \frac{684996955}{339762436}, \frac{28118302103}{267139822}\right]\right]$$

9. Специализированные пакеты *Maple*

Помимо команд, находящихся в основной библиотеке *Maple,* большое количество команд, расширяющих функциональные характеристики программы в отдельных областях математики, находятся в соответствующих специализированных пакетах *Maple.* Некоторые из этих пакетов уже упоминались в других главах. В данном разделе мы кратко перечислим специализированные пакеты и функции, входящие в эти пакеты. Более подробно будут рассмотрены пакеты, не упоминавшиеся в других частях данного пособия. Во многих случаях читатель сможет получить достаточную информацию о функциях по их названиям, однако в некоторых случаях будут даны дополнительные пояснения. Полную информацию можно получить из руководства пользователя или из файлов справки программы.

9.1. *DEtools* — пакет дополнительных средств для дифференциальных уравнений

Этот пакет содержит команды построения двух- и трехмерных графиков решений обыкновенных дифференциальных уравнений и дифференциальных уравнений с частными производными (команды **DEplot**, **DEplot3d**, **и PDEplot**, о которых уже было рассказано в разделах 6 и 7); команды замены переменных в обыкновенных дифференциальных уравнениях (команда **Dchangevar**) и системы координат в уравнениях с частными производными (**PDEchangecoord**); команда понижения порядка дифференциальных уравнений и некоторые другие команды.

9.2. *Domains* — пакет для разработки кодов сложных алгоритмов

Domains — новый пакет *Maple.* Его идея пришла из пакета *AXIOM*, и состоит в том, чтобы передавать в качестве параметра в аргументах процедур набор функций как одно целое, называемое домен (domain). Этот пакет предполагается использовать как средство для разработки сложных алгоритмов. Об этом новом, безусловно перспективном пакете *Maple,* уже подробно рассказано в главе 8.3. Здесь мы только приведем основные команды пакета.

Чтобы загрузить пакет **Domains** необходимо использовать одну из команд **readlib(Domains)**; или **with(Domains)**.

Список всех доменов в настоящее время имеющихся в *Maple* приведен в подразделе 8.3.

Команда **show**(D, operations) может быть использована для вывода на экран всех операций, определенных для домена D.

Для удобства интерактивного использования пакета **Domains** используется команда **evaldomains**[D](expr). **evaldomains** обеспечивает *Maple*-подобный пользовательский интерфейс, компановку данных и проверку типов аргументов **Domains**-функций. Она вычисляет выражение *Maple* в домене D и возвращает выражение *Maple*.

9.3. *GF* — пакет "поля Галуа"

Он содержит только одну команду **GF**(p, k) или **GF**(p, k, a), где

p — простое число;
k — натуральное число;
a — неприводимый полином степени k над полем целых чисел по модулю.

Команда **GF** возвращает таблицу G функций и констант для выполнения арифметических операций над конечным полем **GF**(p^k), то есть полем Галуа с p^k элементами.

9.4. *GaussInt* — пакет Гауссовых целых чисел

Этот пакет содержит около тридцати команд для выполнения операций над полем Гауссовых целых чисел (комплексных чисел с целыми действительной и мнимой частями).

Вот список этих команд:

```
GIbasis     GIchrem     GIdivisor   GIfacpoly     GIfacset
GIfactor    GIfactors   GIgcd       GIgcdex       GIhermite
GIissqr     GIlcm       GImcmbine   GInearest     GInodiv
GInorm      GInormal    GIorder     GIphi         GIprime
GIquadres   GIquo       GIrem       GIroots       GIsieve
GIsmith     GIsqrfree   GIsqrt      GIunitnormal
```

9.5. *LREtools* — пакет для проведения расчетов с рекуррентными соотношениями

Этот пакет дополняет команду **rsolve**, которая находится в основной библиотеке.

Пакет **LREtools** содержит следующий список команд:

REcontent	**REcreate**	**REplot**	**REprimpart**	**REreduceorder**
REtoDE	**REtodelta**	**REtoproc**	**constcoeffsol**	**delta**
dispersion	**hypergeomsols**	**polysols**	**ratpolysols**	**shift**

Рассмотрим пример, в котором рекуррентное уравнение решается как командой **rsolve**, так и при помощи команд пакета **LREtools**, чтобы проиллюстрировать разницу в методах. Отметим, однако, что далеко не все уравнения, решаемые командами **LREtools**, решаются командой **rsolve**.

Пусть задано уравнение

```
> rec := a(n+2)-(2*n+1)*a(n+1)/n+n*a(n)/n-1)=
  n*(n+1):
  rsolve(rec,a(n));
```

$$\left(a(2) + \frac{11}{18}\right)(n - 1) + \frac{1}{9}n^4 - \frac{5}{18}n^3 - \frac{1}{9}n^2 + \frac{5}{18}$$

Так решает рекуррентное уравнение команда **rsolve**.

Теперь воспользуемся пакетом **LREtools**. Вначале необходимо создать структуру данных **RESol** для представления решений рекуррентного уравнения при помощи команды **REcreate**

```
> re1 := LREtools[REcreate](rec, a(n), {});
```

$$re1 := \mathrm{RESol}(\{a(n+2)\,n^2 - a(n+2)\,n - 2a(n+1)\,n^2 + a(n+1) + \\ + n^2\,a(n) = n^4 - n^2\}, \{a(n)\}, \{a(1) = 0, a(3) = a(3), a(2) = a(2)\}, INFO)$$

И только после этого вводится команда вывести решение, например, в виде полинома (возможно также в виде рациональной дроби или гипергеометрической функции).

```
> LREtools[polysols](re1);
```

$$\frac{1}{9}n^4 - \frac{5}{18}n^3 - \frac{1}{9}n^2 + \ a(2) + \frac{11}{18}\ \ n - \frac{1}{3} - a(2)$$

Так решается линейное рекуррентное уравнение при помощи пакета **LREtools**.

9.6. *combinat* — пакет комбинаторики

Он содержит комбинаторные функции, оперирующие с множествами дискретных случайных событий, такими как перестановки, сочетания, разбиения и так далее.

Вот полный список функций пакета:

Chi	**bell**	**binomial**	**cartprod**	**character**
choose	**composition**	**conjpart**	**decodepart**	**encodepart**
fibonacci	**firstpart**	**graycode**	**inttovec**	**lastpart**
multinomial	**nextpart**	**numbcomb**	**numbcomp**	**numbpart**
numbperm	**partition**	**permute**	**powerset**	**prevpart**
randcomb	**randpart**	**randperm**	**stirling1**	**stirling2**
subsets	**vectoint**			

9.7. *combstruct* — пакет комбинаторных структур

Он содержит команды для создания случайно однородных объектов, принадлежащих заданному комбинаторному классу.

Далее полный список этих команд:

allstructs	**count**	**draw**	**finished**	**iterstructs**
nextstruct	**options**	**specification**	**structures**	

Например, для получения всех перестановок списка [x,y,z,t] достаточно записать:

```
> with(combstruct);
  allstructs(Permutation([x,y,z,t]));
```

$$[allstructs,\ count,\ draw,\ finished,\ iterstructs,\ nextstruct]$$

$$[[x, y, z, t], [x, y, t, z], [x, z, y, t], [x, z, t, y], [x, t, y, z], [x, t, z, y],$$
$$[y, x, z, t], [y, x, t, z], [y, z, x, t], [y, z, t, x], [y, t, x, z], [y, t, z, x],$$
$$[z, x, y, t], [z, x, t, y], [z, y, x, t], [z, y, t, x], [z, t, x, y], [z, t, y, x],$$
$$[t, x, y, z], [t, x, z, y], [t, y, x, z], [t, y, z, x], [t, z, x, y], [t, z, y, x]]$$

9.8. *difforms* — пакет дифференциальных форм

Полный список функций пакета:

`const`	`d`	`defform`	`form`	`formpart`
`mixpar`	`parity`	`scalar`	`scalarpart`	`simpform`
`wdegree`	`wedge`			

В следующем примере определена форма и выполнено ее внешнее дифференцирование:

```
with(difforms):
defform(w1=1,w2=1,w3=1,f=scalar, g=scalar,
C=const);
> d( f*w1^2+g*&^(w2,w1)+f*&^(w2,w3) );
```

$$((\mathrm{d}(f))\ \&\hat{}(w1^2)) + \&\hat{}(\mathrm{d}(g), w2, w1) + g((\mathrm{d}(w2))\ \&\hat{}w1) - g(w2\ \&\hat{}(\mathrm{d}(w1))) + \\ + \&\hat{}(\mathrm{d}(f), w2, w3) + f((\mathrm{d}(w2)), w3) - f(w2\ \&\hat{}(\mathrm{d}(w3)))$$

9.9. *finance* — пакет финансовой математики

Он содержит функции для финансовых расчетов.

Следующие функции вычисляют текущую стоимость различных финансовых объектов.

- annuity — для ренты с постоянными выплатами
- cashflows — для выплат, меняющихся от периода к периоду
- growingannuity — для ренты с возрастающими выплатами
- growingperpetuity — бессрочные ренты с возрастающими выплатами
- levelcoupon — для стоимости облигации
- perpetuity — вычисляет текущее значение пожизненной ренты

Другие функции включают

- amortization — таблицы амортизационного списания
- blackscholes — формула Блэка и Фоуэлза
- effectiverate — рост процентной ставки при многократном вложении за период
- futurevalue — будующая величина суммы
- presentvalue — настоящая величина суммы
- yieldtomaturity — доход от ценной бумаги при ее погашении

9.10. *genfunc* — пакет для проведения расчетов с производящими функциями

Рациональные производящие функции, генерируемые командами пакета широко применяются в теории вероятностей.

В пакет входят следующие функции:

rgf_charseq	**rgf_encode**	**rgf_expand**	**rgf_findrecur**	**rgf_hybrid**
rgf_norm	**rgf_pfrac**	**rgf_relate**	**rgf_sequence**	**rgf_simp**
rgf_term	**termscale**			

9.11. *geometry* — геометрический пакет

Функции этого пакета позволяют выполнять построения и вычисления в двухмерной Евклидовой геометрии, а именно создавать различные геометрические фигуры при помощи команд:

circle	**conic**	**dsegment**	**ellipse**	**hyperbola**
line	**parabola**	**point**	**segment**	**square**
triangle				

Пакет позволяет производить различные построения и вычисления, связанные с прямыми, треугольниками, конусами, окружностями, многоугольниками, плоскими кривыми (эллипс, парабола, гипербола), производить различные преобразования (отражения, вращения, растяжения, гомотопии, перемещения, обращения и так далее); команда **draw** позволяет наглядно изобразить все объекты, поддерживаемые пакетом (смотрите подробнее в разделе 6).

9.12. *grobner* — пакет процедур для нахождения базиса Гробнера

Этот пакет – результат внедрения самых современных алгоритмов в программу *Maple*.

Базисом Гробнера Бухбергер назвал по имени руководителя своей докторской диссертации множество многочленов, которое может быть получено из исходного множества многогчленов по алгоритму, разработанному в диссертации Бухбергера. Базис Гробнера – это множество многочленов, в некотором отношении эквивалентное исходному множеству многочленов. Например, корни многочленов базиса Гробнера равны корням исходного множества

многочленов. В то же время коэффициенты многочленов базиса Гробнера составляют верхнюю треугольную матрицу, что позволяет кратчайшим образом найти все корни.

В пакет входят команды:

finduni	**finite**	**gbasis**	**gsolve**	**leadmon**
normalf	**solvable**	**spoly**		

Далее пример использования функций пакета **grobner**. Чтобы найти базис Гробнера набора полиномов F от переменных X = [x1, x2, ..., xn], упорядоченных по полным степеням, нужно вначале задать F и X, а затем выполнить команды, записанные ниже.

```
> with(grobner, gbasis);
  F := [x^2 - 2*x*z + 5, x*y^2 + y*z^3, 3*y^2 -
  8*z^3]:
  gbasis(F,[y,x,z],plex);
```

$$[gbasis]$$

$$[3y^2 - 8\,z^3,\ 80\,y\,z^3 - 3\,z^8 + 32\,z^7 - 40\,z^5,\ x^2 - 2\,x\,z + 5,\ -96\,z^7 + 9\,z^8 + \\ + 120\,z^5 + 640\,z^3\,x,\ 240\,z^6 + 1600\,z^3 - 96\,z^8 + 9\,z^9]$$

9.13. *group* — пакет групп перестановок и конечно-представимых групп

Он содержит следующие функции:

DerivedS	**LCS**	**NormalClosure**	**RandElement**	**Sylow**
areconjugate	**center**	**centralizer**	**convert**	**core**
cosets	**cosrep**	**derived**	**grelgroup**	**groupmember**
grouporder	**inter**	**invperm**	**isabelian**	**isnormal**
issubgroup	**mulperms**	**normalizer**	**orbit**	**permgroup**
permrep	**pres**	**subgrel**	**type**	

Пример вычисления порядка группы перестановок:

```
> with(group):
  grouporder(permgroup(8, {a=[[1,2]],
  b=[[1,2,3,4,5,6,7,8]]}));
```

$$40320$$

9.14. *inttrans* — пакет интегральных преобразований

Он содержит следующие функции для выполнения всех наиболее известных интегральных преобразований:

addtable	**fourier**	**fouriercos**	**fouriersin**	**hankel**
hilbert	**invfourier**	**invhilbert**	**invlaplace**	**laplace**
mellin				

В качестве примера рассмотрим преобразование Лапласа тригонометрической функции

```
> with(inttrans):
  laplace(sin(2*t+3),t,s);
```

$$\frac{s\sin(3)+2\cos(3)}{s^2+4}$$

Обратное преобразование

```
> invlaplace(",s,t);
```

$$\sin(3)\cos(2\,t)+\cos(3)\sin(2\,t)$$

После упрощения получаем исходную функцию

```
> combine(",trig);
```

$$\sin(2\,t+3)$$

9.15. *liesymm* — пакет симметрий Ли

Он применяется для получения определяющих уравнений, при помощи которых решаются системы уравнений в частных производных методом подобия. Пакет включает следующие функции:

&^	**&mod**	**Eta**	**Lie**	**Lrank**
TD	**annul**	**autosimp**	**close**	**d**
depvars	**determine**	**dvalue**	**extvars**	**getcoeff**
getform	**hasclosure**	**hook**	**indepvars**	**makeforms**
mixpar	**prolong**	**reduce**	**setup**	**translate**
vfix	**wcollect**	**wdegree**	**wedgeset**	**wsubs**

Кратко перечислим назначение функций пакета

- **setup** для определения списка координатных переменных (0-forms);
- **d** для вычисления внешних производных по отношению к заданным координатам;
- **&^** для вычисления V-произведения (wedge product);
- **Lie** для вычисления производной Ли выражения, включающего формы по отношению к заданному вектору;
- **wcollect** чтобы выразить форму как сумму форм, каждая из которых умножается на весовой коэффициент;
- **wsubs** чтобы заменить выражение k-формой, являющейся частью n-формы.

Различные другие функции, такие, как **choose**(), **getcoeff**(), **mixpar**(), **wdegree**(), **wedgeset**() и **value**(), используются для операций с формами.

Пример:

```
> with(liesymm):
Warning, new definition for close
> setup();
```

$$[\ ]$$

Задаем дифференциальное уравнение

```
> eq := Diff(u(x,t),x,t) + Diff(u(x,t),x)
  +u(x,t)^2=0;
```

$$eq := \left(\frac{\partial^2}{\partial t\,\partial x}\,\mathrm{u}(x,\,t)\right) + \left(\frac{\partial}{\partial x}\,\mathrm{u}(x,\,t)\right) + \mathrm{u}(x,\,t)^2 = 0$$

Конструируем набор дифференциальных форм из заданного уравнения

```
> forms := makeforms(eq,u(x,t),w);
```

$$forms := [\mathrm{d}(u) - w1\,\mathrm{d}(x) - w2\,\mathrm{d}(t),\, -((\mathrm{d}(t))\ \&\hat{}\ (\mathrm{d}(w2))) + (w1 + u^2)\,((\mathrm{d}(x))\ \&\hat{}\ (\mathrm{d}(t)))]$$

Сортируем переменные в лексикографическом порядке

```
> eq := mixpar(eq);
```

$$eq := \left(\frac{\partial^2}{\partial x\,\partial t}\,\mathrm{u}(x,\,t)\right) + \left(\frac{\partial}{\partial x}\,\mathrm{u}(x,\,t)\right) + \mathrm{u}(x,\,t)^2 = 0$$

Конструируем определяющие уравнения для изовекторов изовариантной группы

```
> determine( eq, V, u(x,t), w ):
> value("):
```

Находим набор координат

```
> wedgeset(0);
```

$$x,\ t,\ u,\ w1,\ w2$$

```
> close(forms):
```

Получаем набор квазилинейных уравнений первого порядка, эквивалентных начальному (eq):

```
> pdes:=annul(", [x,t]);
```

$$pdes := \left[\left(\frac{\partial}{\partial t}\,\mathrm{u}(x,\,t)\right) - \mathrm{w2}(x,\,\mathrm{t}) = 0,\ \left(\frac{\partial}{\partial x}\,\mathrm{u}(x,\,t)\right) - \mathrm{w1}(x,\,\mathrm{t}) = 0,\right.$$

$$\left.\left(\frac{\partial}{\partial x}\,\mathrm{w2}(x,\,t)\right) + \mathrm{w1}(x,\,\mathrm{t}) + \mathrm{u}(x,\,t)^2 = 0,\ \left(\frac{\partial}{\partial t}\,\mathrm{w1}(x,\,t)\right) - \left(\frac{\partial}{\partial x}\,\mathrm{w2}(x,\,t)\right) = 0\right]$$

9.16. *linalg* — пакет линейной алгебры

Он включает обширное количество функций линейной алгебры, часть из которых рассмотрена в подразделе 6.8.

Вот эти функции:

GramSchmidt	**JordanBlock**	**LUdecomp**	**QRdecomp**	**addcol**
addrow	**adjoint**	**angle**	**augment**	**backsub**
band	**basis**	**bezout**	**blockmatrix**	**charmat**
charpoly	**cholesky**	**col**	**coldim**	**colspace**
colspan	**companion**	**cond**	**copyinto**	**crossprod**
curl	**definite**	**delcols**	**delrows**	**det**
diag	**diverge**	**dotprod**	**eigenval**	**eigenvect**
entermatrix	**equal**	**exponential**	**extend**	**ffgausselim**
fibonacci	**forwardsub**	**frobenius**	**gausselim**	**gaussjord**
geneqns	**genmatrix**	**grad**	**hadamard**	**hermite**

hessian	hilbert	htranspose	ihermite	indexfunc
innerprod	intbasis	inverse	ismith	issimilar
iszero	jacobian	jordan	kernel	laplacian
leastsqrs	linsolve	matadd	matrix	minor
minpoly	mulcol	multiply	norm	normalize
orthog	permanent	pivot	potential	randmatrix
randvector	rank	references	row	rowdim
rowspace	rowspan	scalarmul	singval	smith
stack	submatrix	subvector	sumbasis	swapcol
swaprow	sylvester	toeplitz	trace	transpose
vandermonde	vecpotent	vectdim	vector	wronskian

9.17. *logic* — пакет математической логики

Он включает следующие функции, позволяющие оперировать с булевыми выражениями:

bequal	bsimp	canon	convert/MOD2	convert/frominert
convert/toinert	distrib	dual	environ	randbool
satisfy	tautology			

В этом пакете используются следующие булевы операторы: **&and**, **&or**, **¬**, **&iff**, **&nor**, **&nand**, **&xor** и **&implies**.

Например, чтобы упростить булево выражение **(a &and b) &or (a &and (¬ b))**, нужно записать:

```
> with(logic): bsimp((a &and b) &or (a &and
(&not b)));
```

$$a$$

9.18. *networks* — пакет теории графов

Он содержит обширное количество функций, используемых в теории графов:

acycpoly	addedge	addvertex	adjacency	allpairs
ancestor	arrivals	bicomponents	charpoly	chrompoly
complement	complete	components	connect	connectivity
contract	countcuts	counttrees	cube	cycle
cyclebase	daughter	degreeseq	delete	departures
diameter	dinic	djspantree	dodecahedron	draw
duplicate	edges	ends	eweight	flow
flowpoly	fundcyc	getlabel	girth	graph
graphical	gsimp	gunion	head	icosahedron
incidence	incident	indegree	induce	isplanar
maxdegree	mincut	mindegree	neighbors	new
octahedron	outdegree	path	petersen	random
rank	rankpoly	shortpathtree	show	shrink
span	spanpoly	spantree	tail	tetrahedron
tuttepoly	vdegree	vertices	void	vweight

Пример построения графа.

```
> restart;with(networks):
```

Создадаем новый граф g.

```
> new(g):
```

Добавляем вершины 1, 2, .. 6.

```
> addvertex({1,2,3,4,5,6},g);
```

1, 2, 3, 4, 5, 6

Соединяем вершины 1, 2 с вершинами 3, 4, 5, 6.

```
> connect({1,2},{3,4,5,6},g);
```

e1, e2, e3, e4, e5, e6, e7, e8

Строим граф (рис. 76)

```
> draw(g);
```

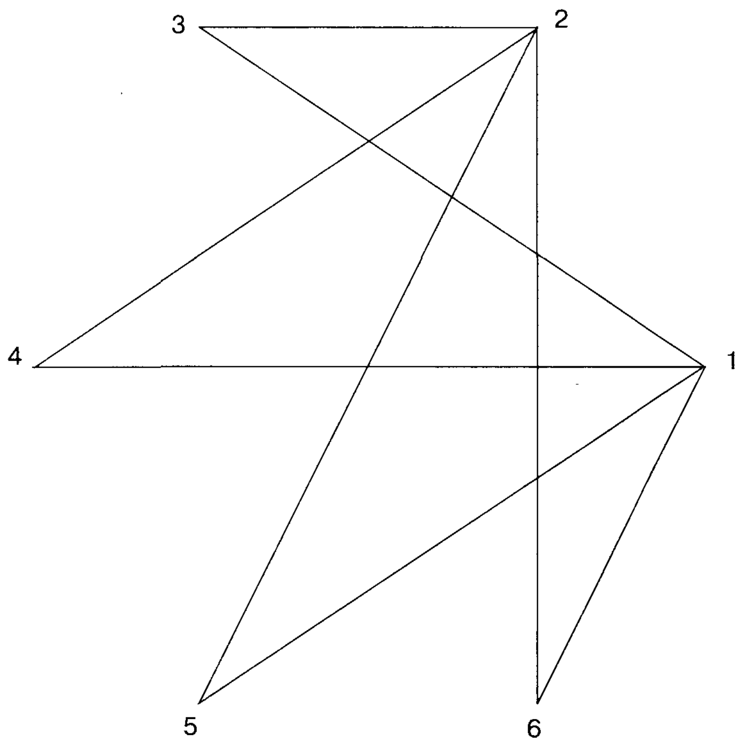

Рис. 76

Связность графа

```
> connectivity(g);
```

$$2$$

Минимальное число разрезов

```
> countcuts(g);
```

$$4$$

9.19. *numapprox* — пакет численной аппроксимации функций

В него входят команды

chebdeg	chebmult	chebpade	chebsort	chebyshev
confracform	hornerform	infnorm	laurent	minimax
pade	remez	taylor		

Например, команда **minimax** предназначена для наилучшего приближения функции на заданном интервале рациональной функцией:

```
> with(numapprox):minimax(sin(x)/x, x=0..2, [2,2]);
```

$$\frac{.9502547411 + (-.0529094941 - .08416376386\, x)\, x}{.9501754310 + (-.05104192415 + .06724432878\, x)\, x}$$

9.20. *numtheory* — пакет теории чисел

Он включает следующие функции:

B	F	GIgcd	J	L
M	bernoulli	bigomega	cfrac	cfracpol
cyclotomic	divisors	euler	factorEQ	factorset
fermat	ifactor	ifactors	imagunit	index
invcfrac	invphi	isolve	isprime	issqrfree
ithprime	jacobi	kronecker	lambda	legendre
mcombine	mersenne	minkowski	mipolys	mlog
mobius	mroot	msqrt	nearestp	nextprime
nthconver	nthdenom	nthnumer	nthpow	order
pdexpand	phi	pprimroot	prevprime	primroot
quadres	rootsunity	safeprime	sigma	sq2factor
sum2sqr	tau	thue		

9.21. *orthopoly* — пакет ортогональных полиномов

В пакет входят следующие функции:

- **G(n, a, x)** создает n-ый полином Гегенбауэра;
- **H(n, x)** создает n-ый полином Эрмита;
- **L(n, x)** создает n-ый полином Лагерра;
- **L(n, a, x)** создает n-ый обобщенный полином Лагерра;
- **P(n, x)** создает n-ый полином Лежандра;
- **P(n, a, b, x)** создает n-ый полином Якоби;
- **T(n, x)** создает n-ый полином Чебышева первого рода;
- **U(n, x)** создает n-ый полином Чебышева второго рода.

```
> with(orthopoly);
Warning, new definition for G
```

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

```
> H(2,x);
```

$$4\,x^2 - 2$$

```
> G(3,a,x);
```

$$\frac{8}{3}\,x^3\,a + 4\,x^3\,a^2 - 2\,a\,x + \frac{4}{3}\,x^3\,a^3 - 2\,x\,a^2$$

9.22. *padic* — пакет для оперирования p-адическими числами

Он содержит следующие функции:

evalp	**expansion**	**function**	**lcoeffp**	**orderp**
ordp	**ratvaluep**	**rootp**	**valuep**	

9.23. *plots* — пакет команд графики и анимации

Он содержит большое количество команд построения двух- и трехмерных графиков. Об этом пакете подробно было рассказано в подразделах 7.1 и 7.2. Здесь приведем только список команд пакета:

animate	**animate3d**	**changecoords**	**complexplot**
complexplot3d	**conformal**	**contourplot**	**contourplot3d**
coordplot	**coordplot3d**	**cylinderplot**	**densityplot**
display	**display3d**	**fieldplot**	**fieldplot3d**
gradplot	**gradplot3d**	**implicitplot**	**implicitplot3d**
inequal	**listcontplot**	**listcontplot3d**	**listdensityplot**
listplot	**listplot3d**	**loglogplot**	**logplot**
matrixplot	**odeplot**	**pareto**	**pointplot**
pointplot3d	**polarplot**	**polygonplot**	**polygonplot3d**
polyhedraplot	**replot**	**rootlocus**	**semilogplot**
setoptions	**setoptions3d**	**spacecurve**	**sparsematrixplot**
sphereplot	**surfdata**	**textplot**	**textplot3d**
tubeplot			

9.24. *plottools* — пакет вспомогательных инструментариев графики

Как уже упоминалось в разделе 7, этот пакет содержит команды, позволяющие создавать графические примитивы для использования в графических структурах. Пакет содержит следующие команды для создания графических объектов:

arc	**arrow**	**circle**	**cone**	**cuboid**
curve	**cutin**	**cutout**	**cylinder**	**disk**
dodecahedron	**ellipse**	**ellipticArc**	**hemisphere**	**hexahedron**
hyperbola	**icosahedron**	**line**	**octahedron**	**pieslice**
point	**polygon**	**rectangle**	**semitorus**	**sphere**
tetrahedron	**torus**			

и следующие команды для преобразования графических объектов:

rotate	**scale**	**stellate**	**transform**	**translate**

9.25. *powseries* — пакет генерации и преобразования степенных рядов

Он содержит функции для генерации степенных рядов и проведения вычислений с ними.

Пакет включает команды:

```
compose     evalpow     inverse     multconst   multiply
negative    powadd      powcos      powcreate   powdiff
powexp      powint      powlog      powpoly     powseries
powsin      powsolve    powsqrt     quotient    reversion
subtract    tpsform
```

Приведем примеры. Командой **powsqrt**(exp(x)) создадим степенной ряд, эквивалентный квадратному корню от экспоненты

```
> with(powseries):
  a := powsqrt(exp(x)):
```

Следующей командой представим этот ряд в виде усеченного степенного ряда

```
> b := powseries[tpsform](a, x, 5);
```

$$b := 1 + \frac{1}{2}x + \frac{1}{8}x^2 + \frac{1}{48}x^3 + \frac{1}{384}x^4 + O(x^5)$$

Команда **powersolve** пакета позволяет находить приближенное решение дифференциальных уравнений в виде степенного ряда

```
> a:=powsolve(diff(y(x),x,x)=y(x),y(0)=1,D(y)(0)=Pi):
  tpsform(a, x);
```

$$1 + \pi x + \frac{1}{2}x^2 + \frac{1}{6}\pi x^3 + \frac{1}{24}x^4 + \frac{1}{120}\pi x^5 + O(x^6)$$

9.26. *simplex* — пакет линейной оптимизации

Он содержит следующие подпрограммы линейной оптимизации, использующие полностью или частично симплекс алгоритм:

```
NONNEGATIVE  basis        convexhull  cterm      define_zero
display      dual         equality    feasible   maximize
minimize     pivot        pivoteqn    pivotvar   ratio
setup        standardize
```

Пример

```
> with(simplex):
Worning, new definition for maximize
Worning, new definition for minimize
```

Вводим ограничения

```
> cnsts := {3*x+4*y-3*z <= 23, 5*x-4*y-3*z <= 10,
  7*x+4*y+11*z <= 30}:
```

и оптимизируемый объект

```
> obj := -x + y + 2*z:
```

теперь находим максимальное значение объекта при заданных ограничениях

```
> maximize(obj,cnsts union {x>=0,y>=0,z>=0});
```

$$\{x = 0, z = \frac{1}{2}, y = \frac{49}{8}\}$$

9.27. *stats* — пакет статистики

Он содержит команды для обработки и проведения статистического анализа данных, а также содержит функции генерации случайных чисел и численного вычисления статистических распределений.

В отличие от других специализированных пакетов этот пакет сложный, то есть он содержит подпакеты, которые в свою очередь содержат команды.

Команда, доступная на верхнем уровне

importdata предназначенная для импортирования данных из файла

Пакет содержит следующие подпакеты:

- **anova** дисперсионный анализ;
- **describe** описательные статистики;
- **fit** линейная регрессия;
- **random** генераторы случайных чисел, соответствующие заданным распределениям;
- **statevalf** численное вычисление распределений;
- **statplots** команды построения графиков;
- **transform** команды преобразования данных.

```
> with(stats):
```

Вводим команду генерации 20 случайных чисел с нормальным распределением

```
> Xdata:=[stats[random, normald](20)];
```

Xdata := [−.8229817284, .2033532865, −.09689410228, 1.059110422, −.5271976236, −.3770007241, .6342440076, 1.610417456, .4759992188, .1262558398, .4062118679, .3585224114, −.4144597255, .2124620638, −.3510011739, .2635298632, −.4225506605, 1.237605096, −1.044155505, 1.001119835]

Преобразуя эти данные при помощи функции sin, получим зависимый набор данных

```
> case:=rand(1000):L:=seq(case()/3000,i=1..20);
```

$$L := \frac{953}{3000}, \frac{431}{3000}, \frac{119}{1000}, \frac{49}{600}, \frac{157}{3000}, \frac{23}{600}, \frac{457}{3000}, \frac{863}{3000}, \frac{109}{3000}, \frac{233}{3000}, \frac{613}{3000}, \frac{4}{15}, \frac{11}{50}, \frac{33}{250}, \frac{193}{100}, \frac{63}{250}, \frac{31}{750}, \frac{167}{3000}, \frac{161}{1500}, \frac{67}{1500}$$

```
> Ydata:=map(sin,Xdata)+[L];
```

Ydata := [−.4155101082, .3456213184, .02225744108, .9535869141, −.4507801158, −.3298001505, .7449020501, 1.286881852, .4945601500, .2035873428, .5994656348, .6175576454, −.1826954576, .3428672401, −.1508381109, .5124901676, −.3687547707, 1.000670099, −.7571670426, .8867421732]

Теперь строим график рассеяния с изображением прямоугольных диаграмм (рис. 77)

```
> plots[display]({
  statplots[scatter2d](Xdata,Ydata)},
  statplots[boxplot[2]](Ydata),
  statplots[xyexchange]
  (statplots[notchedbox[1]](Xdata)),
  view =[-2..2,-1..1], axes=FRAME);
```

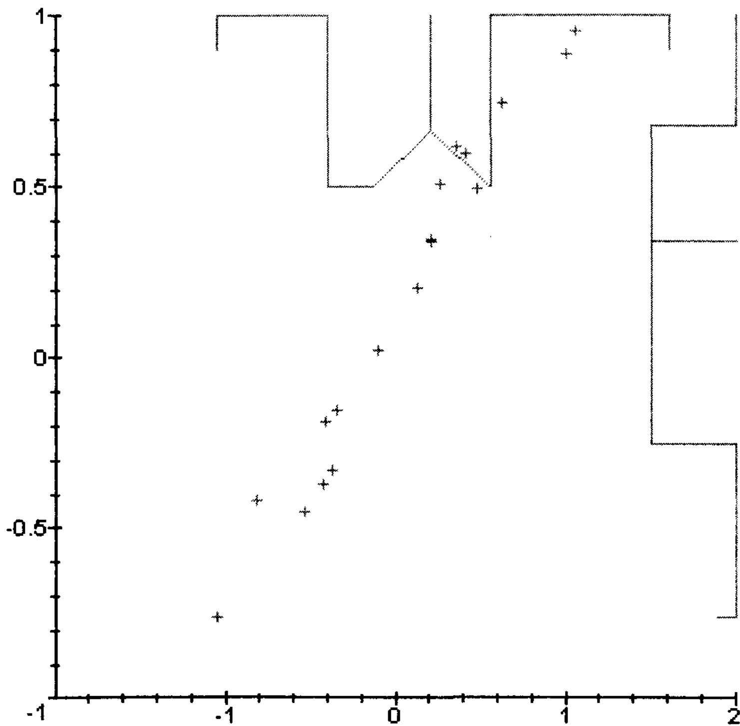

Рис. 77

9.28. *student* — пакет для изучения математики и программирования

Этот пакет, специально написанный для обучения математике и работе с программой. Он содержит набор подпрограмм, предназначенных для выполнения расчетов шаг за шагом, так что студент может понять последовательность действий, приводящих к результату. Интегралы, суммы и пределы приведены в невыполняемой форме. Включены также команды для преобразования выражений, замены переменных, интегрирования по частям, дополнения до полного квадрата и исключения сомножителей. Другие команды вычисляют расстояния, наклоны кривых, средние точки сегментов.

Приведем полный список команд пакета

```
D            Diff         Doubleint        Int          Limit
Lineint      Point        Product          Sum          Tripleint
changevar    combine      completesquare   distance     equate
extrema      integrand    intercept        intparts     isolate
leftbox      leftsum      makeproc         maximize     middlebox
middlesum    midpoint     minimize         minimize     powsubs
rightbox     rightsum     showtangent      simpson      slope
trapezoid    value
```

9.29. *sumtools* — пакет для вычислений конечных и бесконечных сумм

Этот пакет предназначен для вычисления конечных и бесконечных сумм с использованием алгоритма Госпера для бесконечных сумм и алгоритмов Зейлбергера и Копфа.

Пакет содержит следующие команды:

```
Hypersum  Sumtohyper  extended_gosper  gosper  hyperrecursion
hypersum hyperterm  simpcomb  sumrecursion  sumtohyper
```

Приведем пример суммирования рациональной суммы 1/(k^2−1)

```
with(sumtools,gosper):
gosper(1/(k^2-1),k);
```

$$-\frac{1}{2}\frac{(k+1)(-1+2k)}{k(k^2-1)}$$

9.30. *tensor* — пакет тензорной алгебры

Этот пакет новый, в предыдущих версиях его не было, поэтому остановимся на нем подробнее. Пакет содержит команды для оперирования с тензорами и вычислений в рамках в общей теории относительности.

Этот пакет использует свой тип представления данных, называемый **tensor_type**, чтобы представлять объекты как с ковариантными, так и контравариантными индексами. Говоря более конкретно, **tensor_type** — это таблица, содержащая два поля: поле `компонент` для запоминания компонент объекта и поле `характеристик индексов` для описания ковариантных/контравариантных признаков индексов объекта. Поле `компонент` — массив с размерностью, эквивалентной рангу объекта и со всеми диапазонами индексов, начиная с 1 и кончая размерностью пространства (разрешены только диапазоны с одинаковой размерностью для всех индексов). Поле `характеристик индексов` — список положительных и отрицательных единиц. Положительная единица на i-той позиции означает, что i-тый индекс — контравариантный. Аналогично, отрицательная единица означает, что индекс — ковариантный. Например [1, −1, −1, 1] означает, что индексы 1 и 4 контравариантны, а индексы 2 и 3 ковариантны.

tensor_type — процедура, она возвращает **true**, если ее первый аргумент удовлетворяет свойствам тензора и **false** — в противном случае.

Каждому тензору соответствуют еще две таблицы данных: "таблица коэффициентов вращений" и "таблица компонент кривизны".

Таблица коэффициентов вращений запоминает коэффициенты вращения Ньюмена-Пенроуза, вычисляемые командой **tensor[npspin]**. Коэффициенты индексируются в соответствии с их греческими именами: epsilon, nu, lambda, pi, mu, tau, rho, sigma, kappa, alpha, beta и gamma (то есть, элемент таблицы [mu] содержит коэффициент `mu`).

Таблица компонент кривизны содержит компоненты кривизны Ньюмена-Пенроуза, вычисляемые командой **tensor[npcurve]**. Эта таблица содержит три поля: **Phi**, которое является массивом размерности (0..2,0..2) (с эрмитовыми матричными компонентами) и содержит компоненты Риччи; **Psi**, которое является массивом с размерностью (0..4), содержащим компоненты Вейля; и **R**, скалярное поле, содержащее скаляр Риччи.

Пакет содержит также собственную эффективную команду упрощения тензоров **tensor[simp]** и некоторые индексные функции, не содержащиеся в основной библиотеке *Maple*, информацию о которых можно получить при помощи команды **tensor[indexing]**.

Индексные и упрощающие функции пакета инициализируются при помощи **with**[любая команда пакета] либо командой **with(tensor)**.

Приведем полный список команд пакета:

- Christoffel1 — команда вычисления символов Кристоффеля первого рода;
- Christoffel2 — команда вычисления символов Кристоффеля второго рода;

♦ Einstein	тензор Эйнштейна;
♦ display_allGR	описывает ненулевые компоненты всех тензоров и параметров, вычисленных командой tensorsGR (Общая теория относительности);
♦ displayGR	описывает ненулевые компоненты конкретного тензора (Общая теория относительности);
♦ tensorsGR	вычисляет тензор кривизны в данной системе координат (Общая теория относительности);
♦ Jacobian	Якобиан преобразования координат;
♦ Killing_eqns	вычисляет компоненты для уравнений Киллинга (имеет отношение к симметриям пространства);
♦ Levi_Civita	вычисляет ковариантные и контравариантные псевдотензоры Леви-Чивита;
♦ Lie_diff	вычисляет производную Ли тензора по отношению к контравариантному векторному полю;
♦ Ricci	тензор Риччи;
♦ Ricciscalar	скаляр Риччи;
♦ Riemann	тензор Римана;
♦ RiemannF	тензор кривизна Римана в жесткой системе отсчета;
♦ Weyl	тензор Вейля;
♦ act	применяет операции к элементам тензора, таблицам вращений или кривизны;
♦ antisymmetrize	антисимметризация тензора по любым индексам;
♦ change_basis	преобразование системы координат;
♦ commutator	коммутатор двух контравариантных векторных полей;
♦ compare	сравнивает два тензора, таблицы вращений или кривизны;
♦ conj	комплексное сопряжение;
♦ connexF	вычисляет связующие коэффициенты для жесткой системы координат;
♦ contract	свертка тензора по парам индексов;
♦ convertNP	преобразует связующие коэффициенты или тензор Римана к формализму Ньюмена-Пенроуза;
♦ cov_diff	ковариантное дифференцирование;
♦ create	создает тензорный объект;
♦ d1metric	первая частная производная метрики;
♦ d2metric	вторая частная производная метрики;
♦ directional_diff	производная по направлению;

- dual — осуществляет дуальную операцию над индексами тензора;
- entermetric — средство для ввода пользователем координатных переменных и ковариантных компонент метрического тензора
- exterior_diff — внешнее дифференцирование полностью антисимметричного ковариантного тензора;
- exterior_prod — внешнее произведение двух ковариантных антисимметричных тензоров;
- frame — вычисляет систему координат, которая переводит метрические компоненты к диагональной сигнатурной матрице (с положительными или отрицательными единицами);
- geodesic_eqns — уравнение Эйлера-Лагранжа для геодезических кривых;
- get_char — возвращает признак (ковариантный/контравариантный) объекта;
- get_compts — возвращает компоненты объекта;
- get_rank — возвращает ранг объекта;
- invars — инварианты тензора кривизны Римана (Общая теория относительности);
- invert — обращение тензора второго ранга;
- lin_com — линейная комбинация тензорных объектов;
- lower — опускает индексы;
- npcurve — компонента кривизны Ньюмена-Пенроуза в формализме Дебевера (Общая теория относительности);
- npspin — компонента вращения Ньюмена-Пенроуза в формализме Дебевера (Общая теория относительности);
- partial_diff — частная производная тензора;
- permute_indices — перестановка индексов;
- petrov — классификация Петрова тензора Вейля;
- prod — внутреннее и внешнее тензорное произведение;
- raise — поднятие индекса;
- symmetrize — симметризация тензора по любым индексам;
- transform — преобразование системы координат;

Примеры

```
> restart;with(tensor):
```

Введем координаты

```
> coords:=[t, r, theta,phi]:
```

Определим компоненты ковариантного метрического тензора Шварцшильда:

```
> g:=array(symmetric,sparse,1..4,1..4);
  g[1,1]:=(1-2*m/r): g[2,2]:=-1/g[1,1];
  g[3,3]:=-r^2: g[4,4]:=-r^2*sin(theta)^2;
```

$$g := \text{array}(\text{symmetric}, \text{sparse}, 1..4, 1..4, [\,])$$

$$g_{2,2} := -\frac{1}{1-2\,\frac{m}{r}}$$

$$g_{4,4} := -\text{r} \;\; \sin(\text{theta})$$

Теперь создадим метрический тензор metric используя команду create.

```
> metric:=create([-1,-1], eval(g));
```

metric := table([

index_char = [−1, −1]

$$compts = \begin{bmatrix} 1-2\,\frac{m}{r} & 0 & 0 & 0 \\ 0 & -\frac{1}{1-2\,\frac{m}{r}} & 0 & 0 \\ 0 & 0 & -r^2 & 0 \\ 0 & 0 & 0 & -r^2\,\sin(\theta)^2 \end{bmatrix}$$

])

Проверим, что новый объект “metric” действительно тензор ('tensor_type'):

```
> type(metric,tensor_type);
```

true

Вычислим тензоры кривизны по заданному ковариантному метрическому тензору и заданным координатам

```
> tensorsGR(coords,metric,con_metric,det_met, C1,
  C2, Rm, Rc, R, G, C):
```

Здесь con_metric — контравариантный метрический тензор; det_met — детерминант компонент метрического тензора, C1, C2 — символы Кристоффеля 1-го и 2-го рода, Rm -- тензор Римана, Rc — тензор Риччи, R — скаляр Риччи, G — тензор Эйнштейна, C — тензор Вейля.

Теперь при помощи команды displayGR можно вывести на дисплей компоненты любого из перечисленных объектов, указав его имя, которое может быть одним из следующих:

coordinates cov_metric contra_metric detmetric Christoffel1 Christoffel2 Riemann Ricci Ricciscalar Einstein Weyl.

Например

```
> displayGR(Christoffel1,C1);
```

The Christoffel Symbols of the First Kind
non-zero components:

$$[11,2] = -\frac{m}{r^2}$$

$$[12,1] = \frac{m}{r^2}$$

$$[22,2] = -\frac{m}{(r^2-2m)^2}$$

$$[23,3] = -r$$

$$[24,4] = -r\sin(\theta)^2$$

$$[33,2] = r$$

$$[34,4] = -r^2\sin(\theta)\cos(\theta)$$

$$[44,2] = r\sin(\theta)^2$$

$$[44,3] = r^2\sin(\theta)\cos(\theta)$$

$$R3434 = -2\,r\sin(\theta)^2\,m$$

$$C3434 = -2\,r\sin(\theta)^2\,m$$

Для вывода на дисплей тензоров до второго ранга (скаляров, векторов и матриц) удобнее использовать команду **eval**

```
> eval(con_metric);
```

table([

index_char = [1, 1]

$$compts = \begin{bmatrix} \frac{r}{r-2\,m} & 0 & 0 & 0 \\ 0 & -\frac{r-2\,m}{r} & 0 & 0 \\ 0 & 0 & -\frac{1}{r^2} & 0 \\ 0 & 0 & 0 & r^2\,\sin(\theta)^2 \end{bmatrix}$$

])

```
> eval(det_met);
```

$$-r^4\,\sin(\theta)^2$$

9.31. *totorder* — пакет полного упорядочения имен

Пакет содержит команды, вызываемые после загрузки пакета командой **with(totorder)**.

Вот эти команды:

tassume(r) осуществляет полное упорядочение имен по заданной последовательности отношений;

tis(r) функция для осуществления запроса относительно порядка имен;

forget(r) удаляет отношение порядка;

forget(everything) удаляет все отношения порядка;

ordering(r) функция распечатывает текущие отношения порядка;

init(r) функция восстанавливает первоначальное упорядочение,

где r — последовательность соотношений.

Примеры

```
> with(totorder):
> tassume(a<b, b<c, c=d, d<f);
Warning, new definition for init
```

$$assumed,\ a < b,\ b < c,\ c = d,\ d < f$$

```
> tassume(u>b);
```

$$assumed,\ b < u$$

```
> tis(u<f);
```

$$true$$

```
> forget(u);
```

$$forgotten,\ u$$

```
> ordering();
```

$$a < b,\ b < c,\ c = d,\ d < f$$

```
> tassume(u>d);
```

$$assumed,\ d < u$$

```
> tis(2*c-c<f);
```

$$true$$

9.32. Библиотека совместного пользования (share-библиотека)

Это библиотека подпрограмм, пакетов и рабочих документов *Maple*, которые были написаны пользователями этой программы и добровольно предложены сообществу *Maple*.

Share-библиотека распространяется свободно вместе с профессиональной версией *Maple* и располагается в каталоге \share. Доступ к этой библиотеке можно получить командой

```
> with(share);
```

Share-библиотека содержит обширный набор рабочих документов *Maple*. Большинство из них представляют собой решенные в среде *Maple* простые прикладные (инженерные, математические или научные) задачи.

В библиотеке также можно найти подпрограммы и специализированные пакеты вместе с исходными кодами. Во многих случаях можно найти также документацию к пакетам (файлы в формате *LaTeX*) с дополнительными примерами, теоретическим обоснованием, используемыми алгоритмами и так далее.

Программы **share**-библиотеки находятся в следующих подкаталогах, соответствующих различным областям: Algebra, Analysis, Calculus, Combinatorics, Conversions, Courses, Engineering, Geometry, Linear Algebra, Modular computations, Numerics, Number Theory, Graphics, Programming, Science, Statistics, System Tools.

share-библиотека доступна также через *Internet* или по электронной почте. Электронная версия **share**-библиотеки периодически обновляется (примерно трижды в году), поэтому она содержит материалы, которые не включены в стандартно распространяемый комплект *Maple*. Электронная версия **share**-библиотеки содержит также дополнения к библиотеке, исправляющие обнаруженные ошибки программы. Прежде чем использовать команды **share**-библиотеки, необходимо ввести команду:

```
> with(share);
```

Заключение

Прочитав данную книгу, вы получили полное представление о том, что представляет собой программа *Maple V 4.0*, освоили методы проведения аналитических и численных расчетов при ее помощи, а также научились основным приемам программирования. Возможно, вы уже пополнили библиотеки *Maple* собственной библиотекой команд и функций и снабдили ее файлами справки. Вы, несомненно, ощутили комфортность при работе с *Maple*, получили эстетическое удовольствие от формул и графиков, выводимых программой.

Но самое главное, вы почувствовали себя уже почти гением — настолько легко и просто удаются вам сложнейшие расчеты, за которые без этой программы вы бы и не взялись.

Тем не менее, многие читатели, возможно, захотят получить дополнительную информацию об использовании программы в конкретных областях знаний. Для них мы приводим в конце книги список литературы, изданной в последние годы.

Для тех, кто пользуется Internet, мы приводим координаты, адрес электронной почты и www-адрес компании *Waterloo Maple*, где можно найти дополнительную информацию по книгам, учебным пособиям и новым версиям программы:

Maple Waterloo Inc.
450 Phillip Street, Waterloo, ON, Canada N2L 5j2
Phone: (519) 747–2373 Fax: (519) 747–5284
Sales 1-800-267-6583
E-mail: info@Maplesoft.com Web Site: http://www.Maplesoft.com

Хочется надеяться, что прочитанная вами книга в какой-то степени восполнила отсутствие информации в российской печати о самом современном программном обеспечении для автоматизации математических расчетов, среди которого видное место занимает замечательная программа корпорации *Maple Waterloo Maple V Power Edition.*

Автор заранее благодарен всем, приславшим свои замечания как по содержанию книги, так и по обнаруженным ошибкам или опечаткам.

В заключение я хочу поблагодарить Боровикова Игоря, директора корпорации *SoftLine*, инициировавшего написание этой книги и любезно предоставившего программу для работы над книгой.

Эту программу, а также другие математические программы и книги вы можете приобрести в корпорации *SoftLine*.

Справки по телефону 232-00-23.

Литература

Abell, M., J.P. Braselton, *Maple V by Example*, Academic Press, June 1994.

Abell, M., J.P. Braselton, *Differential Equations with Maple V*, Academic Press, September 1994.

Bauldry, W.C., B. Evans, J. Johnson, *Linear Algebra with Maple*, John Wiley & Sons, 1995.

Beltzer, Richard, *Engineering Analysis with Maple/Mathematica*, Academic Press, 1995

Fittahi, A., *Maple V Calculus Labs, Second Edition*, Brooks/Cole, 1995.

*Heal, K.M., M.L. Hansen, K.M. Rickard, *Maple V Learning Guide*, Springer-Verlag, 1996

Horbatsch, M., *Quantum Mechanics Using Maple*, Springer-Verlag, 1995.

Karian, Zaven A., Elliot A. Tanis, *Probability and Statistics Explorations with Maple*, Prentice Hall, 1995.

Komma, Michael, *Moderne Physique mit Maple*, International Thomson Publishing, Nov95.

*Monagan M.B., Geddes K.O., Heal K.M., Labahn G., Vorkoetter S.M., *Maple V. Programming Guide*, Springer Verlag, 1996.

Redfern, Darren, *The Practical Approach, Utilities for Maple V Release 4*, Springer-Verlag, 1997.

Yeargers, E.K., R.W. Shonkwiler, J.V. Herod, *An Introduction to the Mathematics of Biology: With Computer Algebra Models*, Birkhauser, 1997.

Примечание: Книги, помеченные * входят в комплект поставки программы.

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

◆

about 49
act 226
acycpoly 215
add 51
addcol 213
addcoords 97
addedge 215
additionally 91
addrow 213
addvertex 215
adjacency 215
adjoint 213
Algebra 231
allpairs 215
allvalues 123
altitudes 127
ambientlight 129
Analysis 231
ancestor 215
and 214
angle 213
animate 219
animate3d 219
annul 211
anova 222
antisymmetrize 226
arc 219
AreCollinear 128
args 156
array 78
arrivals 215
arrow 219
arrows 109
assign 31
assume 47
augment 213
autosimp 211
axes 223

◆

backsub 213
balloone help 14
band 213
basis 221
bell 207
bequal 214
bernoulli 217
bezout 213
bicomponents 215
bigomega 217
binomial 207
bipolarcylindrical 134
bispherical 134
blockmatrix 213
bsimp 214
by 151

◆

C 229
Calculus 231
canon 214
cardiodal 134
cardiodcylindrical 134
cartprod 207
casscylindrical 134
Catalan 20
center 210
centralizer 210
centroid 127
change_basis 226
changecoords 219
changevar 224
character 207
charmat 213
charpoly 213
chebdeg 217
chebmult 217
chebpade 217
chebsort 217
chebyshev 217
Chi 207
cholesky 213
choose 207
Christoffel 229
Christoffel1 225
Christoffel2 225
chrompoly 215
circle 219
circumcircle 127
close 211
coefficientofvariation 182
col 213
coldim 213
color 97
colspace 213
colspan 213
combinat 207
Combinatorics 231
combine 224
combstruct 207
commutator 226
companion 213
compare 226
complement 215
complete 215
completesquare 224
complex 72
complexplot 219
complexplot3d 219
components 215
composition 207
cond 213
confocalellip 134
confocalparab 134
conformal 219
confracform 217
conic 209
conical 134
conj 226
conjpart 207
connect 215
connectivity 215
connexF 226
const 208
constcoeffsol 206
Contents 16
continuous 74
contourplot 219
contourplot3d 219
contours 107
contract 215, 226
Conversions 231
convert 210
convert/frominert 214
convert/MOD2 214
convert/toinert 214
convertNP 226
convexhull 221
coordplot 219
coordplot3d 219
coordplots 116
copyinto 213
core 210
cosets 210
cosrep 210
count 207
countcuts 215
countmissing 182
counttrees 215
Courses 231
cov_diff 226
covariance 182

create 226
crossprod 213
cterm 221
cube 215
curl 213
cycle 215
cyclebase 215
cylinderplot 219
cylindrical 134

◆

d 208, 211
D 224
d1metric 226
d2metric 226
daughter 215
Dchangevar 204
decile 182
decodepart 207
defform 208
define_zero 221
definite 213
degreeseq 215
delcols 213
delete 215
delrows 213
delta 206
denom 44
DenseUnivariatePolynomial 194
densityplot 219
departures 215
DEplot 204
DEplot3d 204
depvars 211
derived 210
DerivedS 210
describe 222
det 213
determine 211
DEtools 204
dfieldplot 126
diag 213
diameter 215
Diff 224
Diff 40
diff 40, 41
difforms 208
Digits 87
dinic 215
Dirac 46
directional_diff 226
discont 101
disk 219
dispersion 206
display 219, 221
display_allGR 226
display3d 219
displayGR 226
distance 224
distrib 214
ditto 154
diverge 213
divisors 217
djspantree 215
do 151
dodecahedron 215, 219
Domains 204
done 173
dotprod 213
Doubleint 224
draw 207, 215
dsegment 209
dsolve 31
dual 214, 221, 227
duplicate 215
dvalue 211

◆

edges 215
eigenval 213
eigenvals 81
eigenvect 213
eighbors 215
Einstein 226
elif 151
ellipse 209, 219
ellipsoidal 134
ellipticArc 219
EllipticK 75
else 150
encodepart 207
end 152
ends 215
Engineering 231
Enter 10
entermatrix 213
entermetric 227
environ 214
equal 213
equality 221
equate 224
ERROR 152, 158
Eta 211
euler 217
eval 154
evalb 30
evalf 46
evalm 78
evaln 25
evalp 218
evalpow 220
eweight 215
exp(1) 26
expansion 218
exponential 213
extend 213
extended_gosper 224
exterior_diff 227
exterior_prod 227
extrema 224
extvars 211

◆

F 217
factor 48
factorEQ 217
factorset 217
FALSE 124
false 225
feasible 221
fermat 217
ffgausselim 213
fi 150
fibonacci 207, 213
fieldplot 219
fieldplot3d 219
finance 208
finduni 210
finished 207
finite 210
firstpart 207
fit 222
Float 19
float 36
flow 215
flowpoly 215
for 151
forget 230
form 208
formpart 208
fortran 187
forwardsub 213
fourier 211
fouriercos 211
fouriersin 211
fraction 36
frame 227
frames 149
FresnelC 104
frobenius 213
from 151
function 218
fundcyc 215

◆

gamma 20
gausselim 213
GaussInt 205
gaussjord 213
gbasis 210
geneqns 213

genfunc 209
genmatrix 213
geodesic_eqns 227
geometricmean 182
geometry 209
Geometry 231
get_char 227
get_compts 227
get_rank 227
getcoeff 211
getform 211
getlabel 215
GF 205
GIbasis 205
GIchrem 205
GIdivisor 205
GIfacpoly 205
GIfacset 205
GIfactor 205
GIfactors 205
GIgcd 205
GIgcdex 205
GIhermite 205
GIissqr 205
GIlcm 205
GImcmbine 205
GInearest 205
GInodiv 205
GInorm 205
GInormal 205
GIorder 205
GIphi 205
GIprime 205
GIquadres 205
GIquo 205
GIrem 205
GIroots 205
girth 215
GIsieve 205
GIsmith 205
GIsqrfree 205
GIsqrt 205
GIunitnormal 205
global 152
gosper 224
grad 213
gradplot 219
gradplot3d 219
graph 215
graphical 215
Graphics 231
graycode 207
grelgroup 210
grid 95
grobner 209
group 210
groupmember 210
grouporder 210
gsimp 215
gsolve 210
gunion 215

◆

hadamard 213
hankel 211
harmonicmean 182
has 37
hasclosure 211
hastype 37
head 215
hemisphere 219
hermite 213
hessian 214
hexahedron 219
hilbert 211, 214
histogram 120
hook 211
hornerform 217
htranspose 214
hyperbola 209, 219
hypercylindrical 134
hypergeomsols 206
hyperrecursion 224
Hypersum 224
hypersum 224
hyperterm 224

◆

I 20
icosahedron 215, 219
identity 78
if 150, 151
ifactor 217
ifactors 217
ihermite 214
imagunit 217
implicitplot 219
implicitplot3d 219
importdata 221
incidence 215
incident 215
indegree 215
indepvars 211
indet 44
index 217
indexed 22
indexfunc 214
induce 215
inequal 219
infinity 20
infnorm 217
infolevel 135
inits 123
innerprod 214
Int 224
intbasis 214
integer 36
integrand 224
inter 210
intercept 224
interface 175
intersect 27
intparts 224
inttovec 207
inttovec 43
inttrans 211
invars 227
invcasscylindrical 134
invcfrac 217
invellcylindrical 134
inverse 214, 220
invert 227
invfourier 211
invfunc 36
invhilbert 211
invlaplace 211
invoblspheroidal 134
invperm 210
invphi 217
invprospheroidal 134
is 28
isabelian 210
ismith 214
isnormal 210
isolate 224
isolve 217
isplanar 215
isprime 217
issimilar 214
issqrfree 217
issubgroup 210
iszero 214
iterstructs 207
ithprime 217

◆

J 217
jacobi 217
jacobian 214
Jacobian 226
jordan 214
JordanBlock 213

◆

kernel 214
Killing_eqns 226
kronecker 217
kurtosis 182

◆

L 217
labels 143

lambda 217
laplace 211
laplacian 214
lastpart 207
laurent 217
lcoeffp 218
LCS 210
leadmon 210
leastsqrs 214
left 72
leftbox 224
leftsum 224
legendre 217
length 21
Levi_Civita 226
lhs 44
libname 179
Lie 211
Lie_diff 226
liesymm 211
light 129
Limit 224
limit 72
lin_com 227
linalg 213
line 209, 219
linearcorrelation 182
linecolor 126
Lineint 224
linestyle 119
linsolve 214
listcontplot 219
listcontplot3d 219
listdensityplot 219
listplot 219
listplot3d 219
local 71
logcoshcylindrical 134
logcylindrical 134
logic 214
loglogplot 219
logplot 219
lower 227
Lrank 211
LREtools 206
LUdecomp 213

◆

makeforms 211
makeproc 224
map 51
map2 52
matadd 214
matrix 214
matrixplot 219
max 26
maxdegree 215
maximize 221, 224
maxwellcylindrical 134
mcombine 217
mean 41
meandeviation 182
median 182
mersenne 217
method 83
mgear 84
middlebox 224
middlesum 224
midpoint 224
mincut 215
mindegree 215
minimax 217
minimize 221, 224
minkowski 217
minor 214
minpoly 214
minstep 87
minus 27
mipolys 217
mixpar 208, 211
mlog 217
mobius 217
mode 182
moment 182
mroot 217
msqrt 217
mul 51
mulcol 214
mulperms 210
multconst 220
multinomial 207
multiply 214, 220

◆

nearestp 217
networks 215
new 215
nextpart 207
nextprime 217
nops 27, 38
norm 214
NormalClosure 210
normald 222
normalf 210
normalize 214
normalizer 210
not 214
notchedbox 120
npcurve 227
npspin 227
nthconver 217
nthdenom 217
nthnumer 217
nthpow 217
numapprox 217
numbcomb 207
numbcomp 207
Number Theory 231
numbpart 207
numbperm 207
numer 44
numeric 28
Numerics 231
numpoints 97
numtheory 217

◆

oblatespheroidal 134
octahedron 215, 219
od 151
odeplot 219
op 28
options 207
optionsclosed 112
optionsexcluded 112
optionsopen 112
or 214
orbit 210
order 217
ordering 230
orderp 218
ordp 218
orientation 129
orthocenter 127
orthopoly 218
outdegree 215
output 86

◆

padic 218
parabola 209
paraboloidal 134
paraboloidal2 134
paracylindrical 134
pareto 219
parity 208
partial_diff 227
partition 207
path 215
PDEplot 204
pdesolve 89
pdexpand 217
percentile 182
permanent 214
permgroup 210
permrep 210
permute 207
permute_indices 227
petersen 215
petrov 227
phaseportrait 126

phi 217
piecewise 95
pieslice 219
pivot 214, 221
pivoteqn 221
pivotvar 221
plot 13
plot3d 11
plots 219
plottools 219
point 209, 219
Point 224
pointplot 219
pointplot3d 219
polarplot 219
polygon 219
polygonplot 219
polygonplot3d 219
polyhedraplot 219
potential 214
powadd 220
powcos 220
powcreate 220
powdiff 220
powerset 207
powexp 220
powint 220
powlog 220
powpoly 220
powseries 220
powsin 220
powsolve 220
powsqrt 220
powsubs 224
pprimroot 217
precision 185
pres 210
prevpart 207
prevprime 217
primroot 217
print 151
printlevel 157
proc 65
prod 227
Product 224
product 24
Programming 231
prolatespheroidal 134
prolong 211
protect 30

◆

QRdecomp 213
quadraticmean 182
quantile 120
quantile2 120
quartile 182
quit 173
quotient 220

◆

radical 65
radnormal 75
raise 227
randbool 214
randcomb 207
randmatrix 214
random 215, 222
randpart 207
randpart 43
randperm 207
randperm 43
randpoly 62
randvector 214
range 176
rank 214, 215
rankpoly 215
ratio 221
ratpolysols 206
ratvaluep 218
read 176
readlib 205
real 72
REcontent 206
REcreate 206
rectangle 219
reduce 211
references 214
remember 159
remez 217
remove 28
REplot 206
replot 219
REprimpart 206
REreduceorder 206
restart 46
REtoDE 206
REtodelta 206
REtoproc 206
RETURN 152
reversion 220
rgf_charseq 209
rgf_encode 209
rgf_expand 209
rgf_findrecur 209
rgf_hybrid 209
rgf_norm 209
rgf_pfrac 209
rgf_relate 209
rgf_sequence 209
rgf_simp 209
rgf_term 209
rhs 44
Ricci 226
Ricciscalar 226
Riemann 226
RiemannF 226
right 72
rightbox 224
rightsum 224
rootlocus 219
RootOf 62
rootp 218
roots 49
rootsunity 217
rosecylindrical 134
rotate 219
row 214
rowdim 214
rowspace 214
rowspan 214
rsolve 206

◆

safeprime 217
save 176
scalar 208
scalarmul 214
scalarpart 208
scale 219
scatter1d 120
scatter2d 120
scene 123
segment 209
select 28
semilogplot 219
semitorus 219
seq 23
series 54
setoptions 219
setoptions3d 219
setup 211, 212, 221
share 231
shift 206
shortpathtree 215
show 205, 215
showtangent 224
shrink 215
siderel 47
sigma 217
signum 47
simpcomb 224
simpform 208
simplex 221
simplify 47
simpson 224
singval 214
sixsphere 134
skewness 182
slope 224
smith 214

solvable 210
solve 31
sort 28
spacecurve 219
span 215
spanpoly 215
spantree 215
sparse 78
sparsematrixplot 219
specification 207
sphere 219
sphereplot 219
spherical 134
sq2factor 217
sqrt 47
stack 214
standarddeviation 182
standardize 221
statevalf 222
statplots 222
stats 221
stellate 219
stepsize 87
stirling1 207
stirling2 207
stopat 173
stoperror 173
stopwhen 173
string 21
structures 207
student 224
style 98
subgrel 210
submatrix 214
subs 44
subsets 207
subsop 44
substring 22
subvector 214
Sum 224
sum 24
sum2sqr 217
sumbasis 214
sumdata 182
sumrecursion 224
Sumtohyper 224
sumtohyper 224
sumtools 224
surfdata 219
swapcol 214
swaprow 214
Sylow 210
sylvester 214
symmetrize 227
symmetry 120

◆

tail 215
tangentcylindrical 134
tangentsphere 134
tassume 230
tau 217
tautology 214
taylor 217
tensor 225
tensorsGR 226
termscale 209
Testzero 157
tetrahedron 215
tetrahedron 215, 219
textplot 219
textplot3d 219
then 150
thickness 105
thue 217
time 160
tis 230
title 54
to 151
toeplitz 214
toroidal 134
torus 219
totorder 230
tpsform 220
trace 214
transform 219, 222, 227
translate 211, 219
transpose 214
triangle 209
Tripleint 224
TRUE 124
true 20
true 225
tubeplot 219
tuckmarks 129
tuttepoly 215
type 210

◆

unapply 162
union 27
untrace 172

◆

value 224
valuep 218
vandermonde 214
variance 182
vdegree 215
vecpotent 214
vectdim 214
vectoint 207
vector 214
verboseproc 175
vertices 215
view 120
void 215
vweight 215

◆

wcollect 211
wdegree 208, 211
wedge 208
wedgeset 211
Weyl 226
whattype 22
While 151
with 205
writedata 180
wronskian 214
wsubs 211

◆

xtickmarks 98

◆

ytickmarks 98

◆

zip 28

Содержание

1. ЧТО ТАКОЕ MAPLE V ... 8
2. БЫСТРЫЙ СТАРТ ... 10
3. ИНТЕРФЕЙС ... 13
4. ОБЪЕКТЫ MAPLE ... 17
4.1. Язык программы ... 17
4.2. Структура объектов ... 18
Выражения ... 18
Числа и константы, строки и имена ... 19
1. Целые и рациональные числа ... 19
2. Математические константы ... 20
3. Смешивание и совместимость различных типов констант ... 20
4. Строки ... 20
5. Имена ... 21
6. Оператор конкатенации ... 23
7. Использование кавычек в Maple ... 23
Последовательности выражений ... 25
Наборы и списки ... 26
1. Наборы ... 26
2. Оперирование элементами набора (команды **union**, **intersect**, **minus**) ... 27
3. Списки ... 27
4. Оперирование элементами списка (команды **select**, **remove**, **zip**, **sort**) ... 28
Операторы присваивания и уравнения ... 30
Функции ... 32
Операторы Maple ... 35
1. Оператор композиции ... 35
2. Нейтральный оператор ... 36
4.3. Определение типов объектов ... 36
4.4. Анализ структуры объектов ... 38
5. КОМАНДЫ MAPLE ... 40
5.1. Последовательности параметров ... 40
5.2. Как вызвать команду? ... 41
Автоматически загружаемые и загружаемые из библиотек команды ... 42
Команды в пакетах ... 42

5.3. Некоторые часто используемые команды ... 43
Преобразование выражений ... 43
Части выражения (команды **lhs**, **rhs**, **numer**, **denom**, **remove**, **has**, **select**, **indet**, **subs**, **subsop**) ... 44
Команда **simplify** ... 47
Команды **expand** и **factor** ... 48
Команда **normal** ... 49
Команда **combine** ... 49
Команда **assume** ... 49
*Команды **map**, **add**, **mul*** ... 51
*Изменение типа выражения (команда **convert**)* ... 54

6. ПРИМЕРЫ ВЫЧИСЛЕНИЙ ... 56

6.1. Преобразование алгебраических выражений ... 56
Многочлены и рациональные дроби ... 56
Сложные радикалы ... 57
Тригонометрические выражения ... 57
6.2. Решение уравнений и неравенств ... 58
Решение систем уравнений ... 58
Системы линейных уравнений ... 60
Корни многочленов ... 62
Системы нелинейных уравнений ... 64
Решение рекуррентных и функциональных уравнений ... 65
Решение трансцендентных уравнений и систем ... 66
Решение тригонометрических уравнений ... 66
Решение неравенств ... 67
6.3. Нахождение экстремумов функций, симплекс-метод ... 68
6.4. Дифференцирование ... 69
6.5. Пределы ... 72
6.6. Интегрирование ... 73
Аналитическое интегрирование ... 73
Численное интегрирование ... 75
6.7. Суммы и произведения ... 76
6.8. Примеры из линейной алгебры ... 77
Массивы ... 77
Специальные типы матриц ... 78
Управление элементами массивов ... 78
*Команды пакета **linalg*** ... 80
6.9. Обыкновенные дифференциальные уравнения ... 83
6.10. Уравнения в частных производных ... 89

7. ГРАФИКИ И АНИМАЦИЯ В MAPLE ... 97

7.1. Двухмерные графики ... 97
*Графики, построенные при помощи команды **plot*** ... 98
*Графики, построенные при помощи команд пакета **plots*** ... 106
*Графика пакета **plottools*** ... 119
Графика статистического пакета ... 120

*Графика пакета **DEtools*** 123
Графика геометрического пакета 127
7.2. Трехмерные графики и трехмерная анимация 129
*Графики команды **plot3d*** 129
*Построение трехмерных графиков с помощью команд пакета **plots*** 134
*Графика пакета **DEtools*** 145
*Графика пакета **plottools*** 147
Трехмерная анимация 149

8. ПРОГРАММИРОВАНИЕ В СРЕДЕ MAPLE 150

8.1. Процедурное программирование 150
8.1.1. Базисные конструкции языка 150
If/then/else/fi 150
If/then/elif/then/.../else/fi 151
for/from/by/to/do/od 151
While/do/od 151
8.1.2. Процедуры 152
Параметры процедуры 155
Переменные операционной среды 157
Команда прерывания **ERROR** 158
Рекурсивные процедуры, команда **RETURN**, опция **remember** ... 159
Вложенные процедуры 161
Ньютоновская итерация 164
Оператор аффинного преобразования 166
8.1.3. Методы отладки программ 170
Трассировка 170
Отладчик 173
Чтение кодов библиотечных процедур 175
8.1.4. Сохранение процедур и чтение их в сеансе Maple 176
8.1.5. Создание собственной библиотеки и оформление справки по ее командам 176
8.1.6. Чтение и запись данных в файлы 180
Запись данных в файл 180
Чтение данных из файла 181
8.1.7. Перекодировка процедур на языки Си и Фортран 185
8.2. Программирование свойств и правил вычисления функций и операторов 187
*8.2.1. Команда **define*** 187
8.2.2. Программирование правил вычисления 190
8.2.3. Сравнение с шаблоном 192
8.3. Пакет **Domains** 194
*8.3.1. Домены в **Domains*** 194
*8.3.2. Примеры использования пакета **Domains*** 195
8.3.3. Пакет **Domains** в интерактивном режиме 202

9. СПЕЦИАЛИЗИРОВАННЫЕ ПАКЕТЫ MAPLE 204

9.1. **DEtools** — пакет дополнительных средств для дифференциальных уравнений 204

9.2. **Domains** — пакет для разработки кодов сложных алгоритмов 204
9.3. **GF** — пакет "поля Галуа" 205
9.4. **GaussInt** — пакет Гауссовых целых чисел 205
9.5. **LREtools** — пакет для проведения расчетов с рекуррентными соотношениями 206
9.6. **combinat** — пакет комбинаторики 207
9.7. **combstruct** — пакет комбинаторных структур 207
9.8. **difforms** — пакет дифференциальных форм 208
9.9. **finance** — пакет финансовой математики 208
9.10. **genfunc** — пакет для проведения расчетов с производящими функциями 209
9.11. **geometry** — геометрический пакет 209
9.12. **grobner** — пакет процедур для нахождения базиса Гробнера ... 209
9.13. **group** — пакет групп перестановок и конечно-представимых групп 210
9.14. **inttrans** — пакет интегральных преобразований 211
9.15. **liesymm** — пакет симметрий Ли 211
9.16. **linalg** — пакет линейной алгебры 213
9.17. **logic** — пакет математической логики 214
9.18. **networks** — пакет теории графов 215
9.19. **numapprox** — пакет численной аппроксимации функций 217
9.20. **numtheory** — пакет теории чисел 217
9.21. **orthopoly** — пакет ортогональных полиномов 218
9.22. **padic** — пакет для оперирования p-адическими числами 218
9.23. **plots** — пакет команд графики и анимации 219
9.24. **plottools** — пакет вспомогательных инструментариев графики .. 219
9.25. **powseries** — пакет генерации и преобразования степенных рядов 220
9.26. **simplex** — пакет линейной оптимизации 221
9.27. **stats** — пакет статистики 221
9.28. **student** — пакет для изучения математики и программирования 224
9.29. **sumtools** — пакет для вычислений конечных и бесконечных сумм 224
9.30. **tensor** — пакет тензорной алгебры 225
9.31. **totorder** — пакет полного упорядочения имен 230
9.32. Библиотека совместного пользования (**share**-библиотека) 231
ЗАКЛЮЧЕНИЕ 233
ЛИТЕРАТУРА 234
ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 235