

Spreadsheet Functions¹

Contents

List of Financial Functions and Descriptions.....	24
List of Math and Trigonometry Functions and Descriptions	27
List of Statistical Functions and Descriptions	30
ACCRINT function	34
Description.....	34
Syntax.....	34
Remarks.....	35
ACCRINTM function.....	36
Description.....	36
Syntax.....	36
Remarks.....	36
AMORDEGRC function.....	38
Description.....	38
Syntax.....	38
Remarks.....	39
AMORLINC function.....	40
Description.....	40
Syntax.....	40
COUPDAYBS function	41
Description.....	41
Syntax.....	41
Remarks.....	41
COUPDAYS function	43
Description.....	43
Syntax.....	43
Remarks.....	43
COUPDAYSNC function	45
Description.....	45

¹ The spreadsheet software incorporated in the FP&A examinations allows candidates to use functions that work somewhat like those contained in Microsoft Excel. This document provides candidates with additional information (such as descriptions and basic syntax) on Microsoft Excel functions in three categories: Financial Functions, Math and Trigonometry Functions, and Statistical Functions. (Used with permission from Microsoft.)

Syntax.....	45
Remarks.....	45
COUPNCD function.....	47
Description.....	47
Syntax.....	47
Remarks.....	47
COUPNUM function.....	49
Description.....	49
Syntax.....	49
Remarks.....	49
COUPPCD function	51
Description.....	51
Syntax.....	51
Remarks.....	51
CUMIPMT function.....	53
Description.....	53
Syntax.....	53
Remarks.....	53
CUMPRINC function.....	55
Description.....	55
Syntax.....	55
Remarks.....	55
DB function.....	57
Description.....	57
Syntax.....	57
Remarks.....	57
DDB function	59
Description.....	59
Syntax.....	59
Remarks.....	59
DISC function.....	61
Description.....	61
Syntax.....	61
Remarks.....	61
DOLLARDE function	63
Description.....	63
Syntax.....	63

Remarks.....	63
DOLLARFR function.....	64
Description.....	64
Syntax.....	64
Remarks.....	64
DURATION function.....	65
Description.....	65
Syntax.....	65
Remarks.....	66
EFFECT function.....	67
Description.....	67
Syntax.....	67
Remarks.....	67
FV function	68
Description.....	68
Syntax.....	68
Remarks.....	68
FVSCHEDULE function	70
Description.....	70
Syntax.....	70
Remarks.....	70
INTRATE function.....	71
Description.....	71
Syntax.....	71
Remarks.....	71
IPMT function.....	73
Description.....	73
Syntax.....	73
Remarks.....	73
IRR function.....	75
Description.....	75
Syntax.....	75
Remarks.....	76
ISPMT function.....	77
Description.....	77
Syntax.....	77
Remarks.....	77

MDURATION function	78
Description.....	78
Syntax.....	78
Remarks.....	78
MIRR function.....	80
Description.....	80
Syntax.....	80
Remarks.....	80
NOMINAL function	81
Description.....	81
Syntax.....	81
Remarks.....	81
NPER function	82
Description.....	82
Syntax.....	82
NPV function	83
Description.....	83
Syntax.....	83
Remarks.....	83
ODDFPRICE function.....	85
Description.....	85
Syntax.....	85
Remarks.....	86
ODDFYIELD function	89
Description.....	89
Syntax.....	89
Remarks.....	90
ODDLPRICE function.....	91
Description.....	91
Syntax.....	91
Remarks.....	92
ODDLYIELD function	93
Description.....	93
Syntax.....	93
Remarks.....	94
PMT function	96
Description.....	96

Syntax.....	96
Remarks.....	96
PPMT function.....	97
Description.....	97
Syntax.....	97
Remarks.....	97
PRICE function.....	98
Description.....	98
Syntax.....	98
Remarks.....	99
PRICEDISC function.....	100
Description.....	100
Syntax.....	100
Remarks.....	100
PRICEMAT function.....	102
Description.....	102
Syntax.....	102
Remarks.....	102
PV function.....	104
Description.....	104
Syntax.....	104
Remarks.....	105
RATE function.....	106
Description.....	106
Syntax.....	106
Remarks.....	107
RECEIVED function.....	108
Description.....	108
Syntax.....	108
Remarks.....	108
SLN function.....	110
Description.....	110
Syntax.....	110
SYD function.....	111
Description.....	111
Syntax.....	111
Remarks.....	111

TBILLEQ function.....	112
Description.....	112
Syntax.....	112
Remarks.....	112
TBILLPRICE function.....	113
Description.....	113
Syntax.....	113
Remarks.....	113
TBILLYIELD function.....	115
Description.....	115
Syntax.....	115
Remarks.....	115
VDB function.....	117
Description.....	117
Syntax.....	117
XIRR function.....	119
Description.....	119
Syntax.....	119
Remarks.....	119
XNPV function.....	121
Description.....	121
Syntax.....	121
Remarks.....	121
YIELD function.....	123
Description.....	123
Syntax.....	123
Remarks.....	123
YIELDDISC function.....	125
Description.....	125
Syntax.....	125
Remarks.....	125
YIELDMAT function.....	127
Description.....	127
Syntax.....	127
Remarks.....	127
ABS function.....	129
Description.....	129

Syntax.....	129
ACOS function.....	130
Description.....	130
Syntax.....	130
Remark.....	130
ACOSH function.....	131
Description.....	131
Syntax.....	131
AGGREGATE function.....	132
Syntax.....	132
Remarks.....	134
ASIN function.....	136
Description.....	136
Syntax.....	136
Remark.....	136
ASINH function.....	137
Description.....	137
Syntax.....	137
ATAN function.....	138
Description.....	138
Syntax.....	138
Remark.....	138
ATAN2 function.....	139
Description.....	139
Syntax.....	139
Remarks.....	139
ATANH function.....	140
Description.....	140
Syntax.....	140
CEILING function.....	141
Description.....	141
Syntax.....	141
Remarks.....	141
CEILING.PRECISE function.....	142
Description.....	142
Syntax.....	142

Remarks.....	142
COMBIN function.....	143
Description.....	143
Syntax.....	143
Remarks.....	143
COS function	144
Description.....	144
Syntax.....	144
Remark.....	144
COSH function.....	145
Description.....	145
Syntax.....	145
Remark.....	145
DEGREES function	146
Description.....	146
Syntax.....	146
EVEN function.....	147
Description.....	147
Syntax.....	147
Remarks.....	147
EXP function	148
Description.....	148
Syntax.....	148
Remarks.....	148
FACT function.....	149
Description.....	149
Syntax.....	149
FACTDOUBLE function.....	150
Description.....	150
Syntax.....	150
Remarks.....	150
FLOOR function.....	151
Description.....	151
Syntax.....	151
Remarks.....	151
FLOOR.PRECISE function	152

Description.....	152
Syntax.....	152
Remarks.....	152
GCD function.....	153
Description.....	153
Syntax.....	153
Remarks.....	153
INT function.....	154
Description.....	154
Syntax.....	154
LCM function.....	155
Description.....	155
Syntax.....	155
Remarks.....	155
LN function.....	156
Description.....	156
Syntax.....	156
Remark.....	156
LOG function.....	157
Description.....	157
Syntax.....	157
LOG10 function.....	158
Description.....	158
Syntax.....	158
MDETERM function.....	159
Description.....	159
Syntax.....	159
Remarks.....	159
MINVERSE function.....	160
Description.....	160
Syntax.....	160
Remarks.....	160
MMULT function.....	162
Description.....	162
Syntax.....	162
Remarks.....	162
MOD function.....	163

Description.....	163
Syntax.....	163
Remarks.....	163
MROUND function.....	164
Description.....	164
Syntax.....	164
Remark.....	164
MULTINOMIAL function	165
Description.....	165
Syntax.....	165
Remarks.....	165
ODD function	166
Description.....	166
Syntax.....	166
Remarks.....	166
PI function	167
Description.....	167
Syntax.....	167
POWER function.....	168
Description.....	168
Syntax.....	168
Remark.....	168
PRODUCT function.....	169
Description.....	169
Syntax.....	169
QUOTIENT function	170
Description.....	170
Syntax.....	170
Remark.....	170
RADIANS function.....	171
Description.....	171
Syntax.....	171
RAND function	172
Description.....	172
Syntax.....	172
Remarks.....	172
RANDBETWEEN function.....	173

Description.....	173
Syntax.....	173
ROMAN function.....	174
Description.....	174
Syntax.....	174
Remarks.....	174
ROUND function	175
Description.....	175
Syntax.....	175
Remarks.....	175
ROUNDDOWN function	176
Description.....	176
Syntax.....	176
Remarks.....	176
ROUNDUP function.....	177
Description.....	177
Syntax.....	177
Remarks.....	177
SERIESSUM function	178
Description.....	178
Syntax.....	178
Remark	178
SIGN function	179
Description.....	179
Syntax.....	179
SIN function.....	180
Description.....	180
Syntax.....	180
Remark	180
SINH function	181
Description.....	181
Syntax.....	181
Remark	181
SQRT function	182
Description.....	182
Syntax.....	182
Remark	182

SQRTPI function	183
Description.....	183
Syntax.....	183
Remark.....	183
SUBTOTAL function.....	184
Description.....	184
Syntax.....	184
Remarks.....	185
SUM function.....	186
Description.....	186
Syntax.....	186
Remarks.....	186
SUMIF function.....	187
Description.....	187
Syntax.....	187
SUMIFS function.....	189
Description.....	189
Syntax.....	189
Remarks.....	189
SUMPRODUCT function.....	191
Description.....	191
Syntax.....	191
Remarks.....	191
SUMSQ function.....	192
Description.....	192
Syntax.....	192
Remarks.....	192
SUMX2MY2 function.....	193
Description.....	193
Syntax.....	193
Remarks.....	193
SUMX2PY2 function.....	194
Description.....	194
Syntax.....	194
Remarks.....	194
SUMXMY2 function.....	195
Description.....	195

Syntax.....	195
Remarks.....	195
TAN function	196
Description.....	196
Syntax.....	196
Remark.....	196
TANH function.....	197
Description.....	197
Syntax.....	197
Remark.....	197
TRUNC function	198
Description.....	198
Syntax.....	198
Remark.....	198
AVEDEV function.....	199
Description.....	199
Syntax.....	199
Remarks.....	199
AVERAGE function.....	200
Description.....	200
Syntax.....	200
Remarks.....	200
AVERAGEA function	202
Description.....	202
Syntax.....	202
Remarks.....	202
AVERAGEIF function.....	204
Description.....	204
Syntax.....	204
Remarks.....	204
AVERAGEIFS function.....	206
Description.....	206
Syntax.....	206
Remarks.....	206
BETA.DIST function	208
Description.....	208
Syntax.....	208

Remarks.....	208
BETA.INV function.....	209
Description.....	209
Syntax.....	209
Remarks.....	209
BINOM.DIST function.....	210
Description.....	210
Syntax.....	210
Remarks.....	210
BINOM.INV function.....	212
Description.....	212
Syntax.....	212
Remarks.....	212
CHISQ.DIST function.....	213
Description.....	213
Syntax.....	213
Remarks.....	213
CHISQ.DIST.RT function.....	214
Description.....	214
Syntax.....	214
Remarks.....	214
CHISQ.INV function.....	215
Description.....	215
Syntax.....	215
Remarks.....	215
CHISQ.INV.RT function.....	216
Description.....	216
Syntax.....	216
Remarks.....	216
CHISQ.TEST function.....	217
Description.....	217
Syntax.....	217
Remarks.....	217
CONFIDENCE.NORM function.....	219
Description.....	219
Syntax.....	219
Remarks.....	219

CONFIDENCE.T function	221
Description.....	221
Syntax.....	221
Remarks.....	221
CORREL function	222
Description.....	222
Syntax.....	222
Remarks.....	222
COUNT function.....	223
Description.....	223
Syntax.....	223
Remarks.....	223
COUNTA function	225
Description.....	225
Syntax.....	225
Remarks.....	225
COUNTBLANK function	226
Description.....	226
Syntax.....	226
Remark.....	226
COUNTIF function.....	227
Description.....	227
Syntax.....	227
COUNTIFS function.....	228
Description.....	228
Syntax.....	228
Remarks.....	228
COVARIANCE.P function.....	229
Description.....	229
Syntax.....	229
Remarks.....	229
COVARIANCE.S function.....	230
Description.....	230
Syntax.....	230
Remarks.....	230
DEVSQ function.....	231
Description.....	231

Syntax.....	231
Remarks.....	231
EXPON.DIST function	232
Description.....	232
Syntax.....	232
Remarks.....	232
F.DIST function	233
Description.....	233
Syntax.....	233
Remarks.....	233
F.DIST.RT function	234
Description.....	234
Syntax.....	234
Remarks.....	234
F.INV function.....	235
Description.....	235
Syntax.....	235
Remarks.....	235
F.INV.RT function.....	236
Description.....	236
Syntax.....	236
Remarks.....	236
F.TEST function	238
Description.....	238
Syntax.....	238
Remarks.....	238
FISHER function.....	239
Description.....	239
Syntax.....	239
Remarks.....	239
FISHERINV function.....	240
Description.....	240
Syntax.....	240
Remarks.....	240
FORECAST function.....	241
Description.....	241
Syntax.....	241

Remarks.....	241
FREQUENCY function.....	242
Description.....	242
Syntax.....	242
Remarks.....	242
GAMMA.DIST function.....	243
Description.....	243
Syntax.....	243
Remarks.....	243
GAMMA.INV function	245
Description.....	245
Syntax.....	245
Remarks.....	245
GAMMALN function.....	246
Description.....	246
Syntax.....	246
Remarks.....	246
GAMMALN.PRECISE function	247
Description.....	247
Syntax.....	247
Remarks.....	247
GEOMEAN function	248
Description.....	248
Syntax.....	248
Remarks.....	248
GROWTH function.....	249
Description.....	249
Syntax.....	249
Remarks.....	250
HARMEAN function	251
Description.....	251
Syntax.....	251
Remarks.....	251
HYPGEOM.DIST function	252
Syntax.....	252
Remarks.....	252
INTERCEPT function	254

Description.....	254
Syntax.....	254
Remarks.....	254
KURT function	256
Description.....	256
Syntax.....	256
Remarks.....	256
LARGE function	258
Description.....	258
Syntax.....	258
Remarks.....	258
LINEST function	259
Description.....	259
Syntax.....	259
Remarks.....	261
LOGEST function	263
Description.....	263
Syntax.....	263
Remarks.....	264
LOGNORM.DIST function	266
Description.....	266
Syntax.....	266
Remarks.....	266
LOGNORM.INV function.....	267
Description.....	267
Syntax.....	267
Remarks.....	267
MAX function	268
Description.....	268
Syntax.....	268
Remarks.....	268
MAXA function	269
Description.....	269
Syntax.....	269
Remarks.....	269
MEDIAN function	270
Description.....	270

Syntax.....	270
Remarks.....	270
MIN function	272
Description.....	272
Syntax.....	272
Remarks.....	272
MINA function.....	273
Description.....	273
Syntax.....	273
Remarks.....	273
MODE.MULT function.....	274
Description.....	274
Syntax.....	274
Remarks.....	274
MODE.SNGL function.....	275
Description.....	275
Syntax.....	275
Remarks.....	275
NEGBINOM.DIST function	277
Description.....	277
Syntax.....	277
Remarks.....	277
NORM.DIST function.....	279
Description.....	279
Syntax.....	279
Remarks.....	279
NORM.INV function	280
Description.....	280
Syntax.....	280
Remarks.....	280
NORM.S.DIST function.....	281
Description.....	281
Syntax.....	281
Remarks.....	281
NORM.S.INV function	282
Description.....	282
Syntax.....	282

Remarks.....	282
PEARSON function	283
Description.....	283
Syntax.....	283
Remarks.....	283
PERCENTILE.EXC function	284
Description.....	284
Syntax.....	284
Remarks.....	284
PERCENTILE.INC function	285
Description.....	285
Syntax.....	285
Remarks.....	285
PERCENTRANK.EXC function.....	286
Description.....	286
Syntax.....	286
Remarks.....	286
PERCENTRANK.INC function	287
Description.....	287
Syntax.....	287
Remarks.....	287
PERMUT function	288
Description.....	288
Syntax.....	288
Remarks.....	288
POISSON.DIST function	289
Description.....	289
Syntax.....	289
Remarks.....	289
PROB function	291
Description.....	291
Syntax.....	291
Remarks.....	291
QUARTILE.EXC function	292
Description.....	292
Syntax.....	292
Remarks.....	292

QUARTILE.INC function.....	293
Description.....	293
Syntax.....	293
Remarks.....	293
RANK.AVG function	295
Description.....	295
Syntax.....	295
RANK.EQ function	296
Description.....	296
Syntax.....	296
Remarks.....	296
RSQ function	298
Description.....	298
Syntax.....	298
Remarks.....	298
SKEW function.....	300
Description.....	300
Syntax.....	300
Remarks.....	300
SLOPE function.....	301
Description.....	301
Syntax.....	301
Remarks.....	301
SMALL function	303
Description.....	303
Syntax.....	303
Remarks.....	303
STANDARDIZE function	304
Description.....	304
Syntax.....	304
Remarks.....	304
STDEV.P function.....	305
Description.....	305
Syntax.....	305
Remarks.....	305
STDEV.S function.....	307
Description.....	307

Syntax.....	307
Remarks.....	307
STDEVA function	309
Description.....	309
Syntax.....	309
Remarks.....	309
STDEVP function	311
Description.....	311
Syntax.....	311
Remarks.....	311
STEYX function	313
Description.....	313
Syntax.....	313
Remarks.....	313
T.DIST function	315
Description.....	315
Syntax.....	315
Remarks.....	315
T.DIST.2T function	316
Description.....	316
Syntax.....	316
Remarks.....	316
T.DIST.RT function	317
Description.....	317
Syntax.....	317
Remarks.....	317
T.INV function.....	318
Description.....	318
Syntax.....	318
Remarks.....	318
T.INV.2T function.....	319
Description.....	319
Syntax.....	319
Remarks.....	319
TREND function.....	320
Description.....	320
Syntax.....	320

Remarks.....	321
TRIMMEAN function	322
Description.....	322
Syntax.....	322
Remarks.....	322
T.TEST function.....	323
Description.....	323
Syntax.....	323
Remarks.....	323
VAR.P function.....	325
Description.....	325
Syntax.....	325
Remarks.....	325
VAR.S function.....	326
Description.....	326
Syntax.....	326
Remarks.....	326
VARA function	327
Description.....	327
Syntax.....	327
Remarks.....	327
VARPA function.....	329
Description.....	329
Syntax.....	329
Remarks.....	329
WEIBULL.DIST function	331
Description.....	331
Syntax.....	331
Remarks.....	331
Z.TEST function.....	332
Description.....	332
Syntax.....	332
Remarks.....	332

List of Financial Functions and Descriptions

Function	Description
ACCRINT function	Returns the accrued interest for a security that pays periodic interest
ACCRINTM function	Returns the accrued interest for a security that pays interest at maturity
AMORDEGRC function	Returns the depreciation for each accounting period by using a depreciation coefficient
AMORLINC function	Returns the depreciation for each accounting period
COUPDAYBS function	Returns the number of days from the beginning of the coupon period to the settlement date
COUPDAYS function	Returns the number of days in the coupon period that contains the settlement date
COUPDAYSNC function	Returns the number of days from the settlement date to the next coupon date
COUPNCD function	Returns the next coupon date after the settlement date
COUPNUM function	Returns the number of coupons payable between the settlement date and maturity date
COUPPCD function	Returns the previous coupon date before the settlement date
CUMIPMT function	Returns the cumulative interest paid between two periods
CUMPRINC function	Returns the cumulative principal paid on a loan between two periods
DB function	Returns the depreciation of an asset for a specified period by using the fixed-declining balance method
DDB function	Returns the depreciation of an asset for a specified period by using the double-declining balance method or some other method that you specify
DISC function	Returns the discount rate for a security
DOLLARDE function	Converts a dollar price, expressed as a fraction, into a dollar price, expressed as a decimal number
DOLLARFR function	Converts a dollar price, expressed as a decimal number, into a dollar price, expressed as a fraction
DURATION function	Returns the annual duration of a security with periodic interest payments
EFFECT function	Returns the effective annual interest rate
FV function	Returns the future value of an investment
FVSCHEDULE function	Returns the future value of an initial principal after applying a series of compound interest rates
INTRATE function	Returns the interest rate for a fully invested security
IPMT function	Returns the interest payment for an investment for a given period

IRR function	Returns the internal rate of return for a series of cash flows
ISPMT function	Calculates the interest paid during a specific period of an investment
MDURATION function	Returns the Macauley modified duration for a security with an assumed par value of \$100
MIRR function	Returns the internal rate of return where positive and negative cash flows are financed at different rates
NOMINAL function	Returns the annual nominal interest rate
NPER function	Returns the number of periods for an investment
NPV function	Returns the net present value of an investment based on a series of periodic cash flows and a discount rate
ODDFPRICE function	Returns the price per \$100 face value of a security with an odd first period
ODDFYIELD function	Returns the yield of a security with an odd first period
ODDLPRICE function	Returns the price per \$100 face value of a security with an odd last period
ODDLYIELD function	Returns the yield of a security with an odd last period
PMT function	Returns the periodic payment for an annuity
PPMT function	Returns the payment on the principal for an investment for a given period
PRICE function	Returns the price per \$100 face value of a security that pays periodic interest
PRICEDISC function	Returns the price per \$100 face value of a discounted security
PRICEMAT function	Returns the price per \$100 face value of a security that pays interest at maturity
PV function	Returns the present value of an investment
RATE function	Returns the interest rate per period of an annuity
RECEIVED function	Returns the amount received at maturity for a fully invested security
SLN function	Returns the straight-line depreciation of an asset for one period
SYD function	Returns the sum-of-years' digits depreciation of an asset for a specified period
TBILLEQ function	Returns the bond-equivalent yield for a Treasury bill
TBILLPRICE function	Returns the price per \$100 face value for a Treasury bill
TBILLYIELD function	Returns the yield for a Treasury bill
VDB function	Returns the depreciation of an asset for a specified or partial period by using a declining balance method
XIRR function	Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic
XNPV function	Returns the net present value for a schedule of cash flows that is not necessarily periodic

YIELD function	Returns the yield on a security that pays periodic interest
YIELDDISC function	Returns the annual yield for a discounted security; for example, a Treasury bill
YIELDMAT function	Returns the annual yield of a security that pays interest at maturity

List of Math and Trigonometry Functions and Descriptions

Function	Description
ABS function	Returns the absolute value of a number
ACOS function	Returns the arccosine of a number
ACOSH function	Returns the inverse hyperbolic cosine of a number
AGGREGATE function	Returns an aggregate in a list or database
ASIN function	Returns the arcsine of a number
ASINH function	Returns the inverse hyperbolic sine of a number
ATAN function	Returns the arctangent of a number
ATAN2 function	Returns the arctangent from x- and y-coordinates
ATANH function	Returns the inverse hyperbolic tangent of a number
CEILING function	Rounds a number to the nearest integer or to the nearest multiple of significance
CEILING.PRECISE function	Rounds a number the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded up.
COMBIN function	Returns the number of combinations for a given number of objects
COS function	Returns the cosine of a number
COSH function	Returns the hyperbolic cosine of a number
DEGREES function	Converts radians to degrees
EVEN function	Rounds a number up to the nearest even integer
EXP function	Returns e raised to the power of a given number
FACT function	Returns the factorial of a number
FACTDOUBLE function	Returns the double factorial of a number
FLOOR function	Rounds a number down, toward zero
FLOOR.PRECISE function	Rounds a number down to the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded down.
GCD function	Returns the greatest common divisor
INT function	Rounds a number down to the nearest integer
LCM function	Returns the least common multiple

LN function	Returns the natural logarithm of a number
LOG function	Returns the logarithm of a number to a specified base
LOG10 function	Returns the base-10 logarithm of a number
MDETERM function	Returns the matrix determinant of an array
MINVERSE function	Returns the matrix inverse of an array
MMULT function	Returns the matrix product of two arrays
MOD function	Returns the remainder from division
MROUND function	Returns a number rounded to the desired multiple
MULTINOMIAL function	Returns the multinomial of a set of numbers
ODD function	Rounds a number up to the nearest odd integer
PI function	Returns the value of pi
POWER function	Returns the result of a number raised to a power
PRODUCT function	Multiplies its arguments
QUOTIENT function	Returns the integer portion of a division
RADIANS function	Converts degrees to radians
RAND function	Returns a random number between 0 and 1
RANDBETWEEN function	Returns a random number between the numbers you specify
ROMAN function	Converts an arabic numeral to roman, as text
ROUND function	Rounds a number to a specified number of digits
ROUNDDOWN function	Rounds a number down, toward zero
ROUNDUP function	Rounds a number up, away from zero
SERIESSUM function	Returns the sum of a power series based on the formula
SIGN function	Returns the sign of a number
SIN function	Returns the sine of the given angle
SINH function	Returns the hyperbolic sine of a number
SQRT function	Returns a positive square root
SQRTPI function	Returns the square root of (number * pi)

SUBTOTAL function	Returns a subtotal in a list or database
SUM function	Adds its arguments
SUMIF function	Adds the cells specified by a given criteria
SUMIFS function	Adds the cells in a range that meet multiple criteria
SUMPRODUCT function	Returns the sum of the products of corresponding array components
SUMSQ function	Returns the sum of the squares of the arguments
SUMX2MY2 function	Returns the sum of the difference of squares of corresponding values in two arrays
SUMX2PY2 function	Returns the sum of the sum of squares of corresponding values in two arrays
SUMXMY2 function	Returns the sum of squares of differences of corresponding values in two arrays
TAN function	Returns the tangent of a number
TANH function	Returns the hyperbolic tangent of a number
TRUNC function	Truncates a number to an integer

List of Statistical Functions and Descriptions

Function	Description
AVEDEV function	Returns the average of the absolute deviations of data points from their mean
AVERAGE function	Returns the average of its arguments
AVERAGEA function	Returns the average of its arguments, including numbers, text, and logical values
AVERAGEIF function	Returns the average (arithmetic mean) of all the cells in a range that meet a given criteria
AVERAGEIFS function	Returns the average (arithmetic mean) of all cells that meet multiple criteria
BETA.DIST function	Returns the beta cumulative distribution function
BETA.INV function	Returns the inverse of the cumulative distribution function for a specified beta distribution
BINOM.DIST function	Returns the individual term binomial distribution probability
BINOM.INV function	Returns the smallest value for which the cumulative binomial distribution is less than or equal to a criterion value
CHISQ.DIST function	Returns the cumulative beta probability density function
CHISQ.DIST.RT function	Returns the one-tailed probability of the chi-squared distribution
CHISQ.INV function	Returns the cumulative beta probability density function
CHISQ.INV.RT function	Returns the inverse of the one-tailed probability of the chi-squared distribution
CHISQ.TEST function	Returns the test for independence
CONFIDENCE.NORM function	Returns the confidence interval for a population mean
CONFIDENCE.T function	Returns the confidence interval for a population mean, using a Student's t distribution
CORREL function	Returns the correlation coefficient between two data sets
COUNT function	Counts how many numbers are in the list of arguments
COUNTA function	Counts how many values are in the list of arguments
COUNTBLANK function	Counts the number of blank cells within a range
COUNTIF function	Counts the number of cells within a range that meet the given criteria
COUNTIFS function	Counts the number of cells within a range that meet multiple criteria
COVARIANCE.P function	Returns covariance, the average of the products of paired deviations
COVARIANCE.S function	Returns the sample covariance, the average of the products deviations for each data point pair in two data sets

DEVSQ function	Returns the sum of squares of deviations
EXPON.DIST function	Returns the exponential distribution
F.DIST function	Returns the F probability distribution
F.DIST.RT function	Returns the F probability distribution
F.INV function	Returns the inverse of the F probability distribution
F.INV.RT function	Returns the inverse of the F probability distribution
F.TEST function	Returns the result of an F-test
FISHER function	Returns the Fisher transformation
FISHERINV function	Returns the inverse of the Fisher transformation
FORECAST function	Returns a value along a linear trend
FREQUENCY function	Returns a frequency distribution as a vertical array
GAMMA.DIST function	Returns the gamma distribution
GAMMA.INV function	Returns the inverse of the gamma cumulative distribution
GAMMALN function	Returns the natural logarithm of the gamma function, $\Gamma(x)$
GAMMALN.PRECISE function	Returns the natural logarithm of the gamma function, $\Gamma(x)$
GEOMEAN function	Returns the geometric mean
GROWTH function	Returns values along an exponential trend
HARMEAN function	Returns the harmonic mean
HYPGEOM.DIST function	Returns the hypergeometric distribution
INTERCEPT function	Returns the intercept of the linear regression line
KURT function	Returns the kurtosis of a data set
LARGE function	Returns the k-th largest value in a data set
LINEST function	Returns the parameters of a linear trend
LOGEST function	Returns the parameters of an exponential trend
LOGNORM.DIST function	Returns the cumulative lognormal distribution
LOGNORM.INV function	Returns the inverse of the lognormal cumulative distribution
MAX function	Returns the maximum value in a list of arguments
MAXA function	Returns the maximum value in a list of arguments, including numbers, text, and logical values

MEDIAN function	Returns the median of the given numbers
MIN function	Returns the minimum value in a list of arguments
MINA function	Returns the smallest value in a list of arguments, including numbers, text, and logical values
MODE.MULT function	Returns a vertical array of the most frequently occurring, or repetitive values in an array or range of data
MODE.SNGL function	Returns the most common value in a data set
NEGBINOM.DIST function	Returns the negative binomial distribution
NORM.DIST function	Returns the normal cumulative distribution
NORM.INV function	Returns the inverse of the normal cumulative distribution
NORM.S.DIST function	Returns the standard normal cumulative distribution
NORM.S.INV function	Returns the inverse of the standard normal cumulative distribution
PEARSON function	Returns the Pearson product moment correlation coefficient
PERCENTILE.EXC function	Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive
PERCENTILE.INC function	Returns the k-th percentile of values in a range
PERCENTRANK.EXC function	Returns the rank of a value in a data set as a percentage (0..1, exclusive) of the data set
PERCENTRANK.INC function	Returns the percentage rank of a value in a data set
PERMUT function	Returns the number of permutations for a given number of objects
POISSON.DIST function	Returns the Poisson distribution
PROB function	Returns the probability that values in a range are between two limits
QUARTILE.EXC function	Returns the quartile of the data set, based on percentile values from 0..1, exclusive
QUARTILE.INC function	Returns the quartile of a data set
RANK.AVG function	Returns the rank of a number in a list of numbers
RANK.EQ function	Returns the rank of a number in a list of numbers
RSQ function	Returns the square of the Pearson product moment correlation coefficient
SKEW function	Returns the skewness of a distribution
SLOPE function	Returns the slope of the linear regression line
SMALL function	Returns the k-th smallest value in a data set
STANDARDIZE function	Returns a normalized value

STDEV.P function	Calculates standard deviation based on the entire population
STDEV.S function	Estimates standard deviation based on a sample
STDEVA function	Estimates standard deviation based on a sample, including numbers, text, and logical values
STDEVPA function	Calculates standard deviation based on the entire population, including numbers, text, and logical values
STEYX function	Returns the standard error of the predicted y-value for each x in the regression
T.DIST function	Returns the Percentage Points (probability) for the Student t-distribution
T.DIST.2T function	Returns the Percentage Points (probability) for the Student t-distribution
T.DIST.RT function	Returns the Student's t-distribution
T.INV function	Returns the t-value of the Student's t-distribution as a function of the probability and the degrees of freedom
T.INV.2T function	Returns the inverse of the Student's t-distribution
TREND function	Returns values along a linear trend
TRIMMEAN function	Returns the mean of the interior of a data set
T.TEST function	Returns the probability associated with a Student's t-test
VAR.P function	Calculates variance based on the entire population
VAR.S function	Estimates variance based on a sample
VARA function	Estimates variance based on a sample, including numbers, text, and logical values
VARPA function	Calculates variance based on the entire population, including numbers, text, and logical values
WEIBULL.DIST function	Returns the Weibull distribution
Z.TEST function	Returns the one-tailed probability-value of a z-test

ACCRINT function

Description

Returns the accrued interest for a security that pays periodic interest.

Syntax

```
ACCRINT(issue, first_interest, settlement, rate, par, frequency, [basis],  
[calc_method])
```

The ACCRINT function syntax has the following arguments:

- **Issue** Required. The security's issue date.
- **First_interest** Required. The security's first interest date.
- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Rate** Required. The security's annual coupon rate.
- **Par** Required. The security's par value. If you omit par, ACCRINT uses \$1,000.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

- **Calc_method** Optional. A logical value that specifies the way to calculate the total accrued interest when the date of settlement is later than the date of first_interest. A value of TRUE (1)

returns the total accrued interest from issue to settlement. A value of FALSE (0) returns the accrued interest from first_interest to settlement. If you do not enter the argument, it defaults to TRUE.

Remarks

- Issue, first_interest, settlement, frequency, and basis are truncated to integers.
- If issue, first_interest, or settlement is not a valid date, ACCRINT returns the #VALUE! error value.
- If rate ≤ 0 or if par ≤ 0 , ACCRINT returns the #NUM! error value.
- If frequency is any number other than 1, 2, or 4, ACCRINT returns the #NUM! error value.
- If basis < 0 or if basis > 4 , ACCRINT returns the #NUM! error value.
- If issue \geq settlement, ACCRINT returns the #NUM! error value.
- ACCRINT is calculated as follows:

$$ACCRINT = par \times \frac{rate}{frequency} \times \sum_{i=1}^{NC} \frac{A_i}{NL_i}$$

where:

- A_i = number of accrued days for the i th quasi-coupon period within odd period.
- NC = number of quasi-coupon periods that fit in odd period. If this number contains a fraction, raise it to the next whole number.
- NL_i = normal length in days of the i th quasi-coupon period within odd period.

ACCRINTM function

Description

Returns the accrued interest for a security that pays interest at maturity.

Syntax

```
ACCRINTM(issue, settlement, rate, par, [basis])
```

The ACCRINTM function syntax has the following arguments:

- **Issue** Required. The security's issue date.
- **Settlement** Required. The security's maturity date.
- **Rate** Required. The security's annual coupon rate.
- **Par** Required. The security's par value. If you omit par, ACCRINTM uses \$1,000.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- Issue, settlement, and basis are truncated to integers.
- If issue or settlement is not a valid date, ACCRINTM returns the #VALUE! error value.
- If rate ≤ 0 or if par ≤ 0 , ACCRINTM returns the #NUM! error value.
- If basis < 0 or if basis > 4 , ACCRINTM returns the #NUM! error value.

- If issue \geq settlement, ACCRINTM returns the #NUM! error value.
- ACCRINTM is calculated as follows:

$$ACCRINTM = par \times rate \times \frac{A}{D}$$

where:

- A = Number of accrued days counted according to a monthly basis. For interest at maturity items, the number of days from the issue date to the maturity date is used.
- D = Annual Year Basis.

AMORDEGRC function

Description

Returns the depreciation for each accounting period. This function is provided for the French accounting system. If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account. The function is similar to AMORLINC, except that a depreciation coefficient is applied in the calculation depending on the life of the assets.

Syntax

```
AMORDEGRC(cost, date_purchased, first_period, salvage, period, rate, [basis])
```

IMPORTANT Dates should be entered by using the DATE function, or as results of other formulas or functions. For example, use DATE(2008,5,23) for the 23rd day of May, 2008. Problems can occur if dates are entered as text.

The AMORDEGRC function syntax has the following arguments:

- **Cost** Required. The cost of the asset.
- **Date_purchased** Required. The date of the purchase of the asset.
- **First_period** Required. The date of the end of the first period.
- **Salvage** Required. The salvage value at the end of the life of the asset.
- **Period** Required. The period.
- **Rate** Required. The rate of depreciation.
- **Basis** Optional. The year basis to be used.

Basis	Date system
0 or omitted	360 days (NASD method)
1	Actual
3	365 days in a year
4	360 days in a year (European method)

Remarks

- This function will return the depreciation until the last period of the life of the assets or until the cumulated value of depreciation is greater than the cost of the assets minus the salvage value.
- The depreciation coefficients are:

Life of assets (1/rate)	Depreciation coefficient
Between 3 and 4 years	1.5
Between 5 and 6 years	2
More than 6 years	2.5

- The depreciation rate will grow to 50 percent for the period preceding the last period and will grow to 100 percent for the last period.
- If the life of assets is between 0 (zero) and 1, 1 and 2, 2 and 3, or 4 and 5, the #NUM! error value is returned.

AMORLINC function

Description

Returns the depreciation for each accounting period. This function is provided for the French accounting system. If an asset is purchased in the middle of the accounting period, the prorated depreciation is taken into account.

Syntax

```
AMORLINC(cost, date_purchased, first_period, salvage, period, rate, [basis])
```

The AMORLINC function syntax has the following arguments:

- **Cost** Required. The cost of the asset.
- **Date_purchased** Required. The date of the purchase of the asset.
- **First_period** Required. The date of the end of the first period.
- **Salvage** Required. The salvage value at the end of the life of the asset.
- **Period** Required. The period.
- **Rate** Required. The rate of depreciation.
- **Basis** Optional. The year basis to be used.

Basis	Date system
0 or omitted	360 days (NASD method)
1	Actual
3	365 days in a year
4	360 days in a year (European method)

COUPDAYBS function

Description

Returns the number of days in the coupon period that contains the settlement date.

Syntax

```
COUPDAYBS(settlement, maturity, frequency, [basis])
```

The COUPDAYBS function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008,

the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, 30 years after the January 1, 2008, issue date.

- All arguments are truncated to integers.
- If settlement or maturity is not a valid date, COUPDAYBS returns the #VALUE! error value.
- If frequency is any number other than 1, 2, or 4, COUPDAYBS returns the #NUM! error value.
- If basis < 0 or if basis > 4, COUPDAYBS returns the #NUM! error value.
- If settlement \geq maturity, COUPDAYBS returns the #NUM! error value.

COUPDAYS function

Description

Returns the number of days in the coupon period that contains the settlement date.

Syntax

```
COUPDAYS(settlement, maturity, frequency, [basis])
```

The COUPDAYS function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, 30 years after the January 1, 2008 issue date.

- All arguments are truncated to integers.
- If settlement or maturity is not a valid date, COUPDAYS returns the #VALUE! error value.
- If frequency is any number other than 1, 2, or 4, COUPDAYS returns the #NUM! error value.
- If basis < 0 or if basis > 4, COUPDAYS returns the #NUM! error value.
- If settlement \geq maturity, COUPDAYS returns the #NUM! error value.

COUPDAYSNC function

Description

Returns the number of days from the settlement date to the next coupon date.

Syntax

```
COUPDAYSNC(settlement, maturity, frequency, [basis])
```

The COUPDAYSNC function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- All arguments are truncated to integers.
- If settlement or maturity is not a valid date, COUPDAYSNC returns the #VALUE! error value.
- If frequency is any number other than 1, 2, or 4, COUPDAYSNC returns the #NUM! error value.
- If basis < 0 or if basis > 4, COUPDAYSNC returns the #NUM! error value.
- If settlement \geq maturity, COUPDAYSNC returns the #NUM! error value.

COUPNCD function

Description

Returns a number that represents the next coupon date after the settlement date.

Syntax

```
COUPNCD(settlement, maturity, frequency, [basis])
```

The COUPNCD function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- All arguments are truncated to integers.
- If settlement or maturity is not a valid date, COUPNCD returns the #VALUE! error value.
- If frequency is any number other than 1, 2, or 4, COUPNCD returns the #NUM! error value.
- If basis < 0 or if basis > 4, COUPNCD returns the #NUM! error value.
- If settlement \geq maturity, COUPNCD returns the #NUM! error value.

COUPNUM function

Description

Returns the number of coupons payable between the settlement date and maturity date, rounded up to the nearest whole coupon.

Syntax

```
COUPNUM(settlement, maturity, frequency, [basis])
```

The COUPNUM function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- All arguments are truncated to integers.
- If settlement or maturity is not a valid date, COUPNUM returns the #VALUE! error value.
- If frequency is any number other than 1, 2, or 4, COUPNUM returns the #NUM! error value.
- If basis < 0 or if basis > 4, COUPNUM returns the #NUM! error value.
- If settlement \geq maturity, COUPNUM returns the #NUM! error value.

COUPPCD function

Description

Returns a number that represents the previous coupon date before the settlement date.

Syntax

```
COUPPCD(settlement, maturity, frequency, [basis])
```

IMPORTANT Dates should be entered by using the DATE function, or as results of other formulas or functions. For example, use DATE(2008,5,23) for the 23rd day of May, 2008. Problems can occur if dates are entered as text.

The COUPPCD function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- All arguments are truncated to integers.
- If settlement or maturity is not a valid date, COUPPCD returns the #VALUE! error value.
- If frequency is any number other than 1, 2, or 4, COUPPCD returns the #NUM! error value.
- If basis < 0 or if basis > 4, COUPPCD returns the #NUM! error value.
- If settlement \geq maturity, COUPPCD returns the #NUM! error value.

CUMIPMT function

Description

Returns the cumulative interest paid on a loan between start_period and end_period.

Syntax

```
CUMIPMT(rate, nper, pv, start_period, end_period, type)
```

The CUMIPMT function syntax has the following arguments:

- **Rate** Required. The interest rate.
- **Nper** Required. The total number of payment periods.
- **Pv** Required. The present value.
- **Start_period** Required. The first period in the calculation. Payment periods are numbered beginning with 1.
- **End_period** Required. The last period in the calculation.
- **Type** Required. The timing of the payment.

Type	Timing
0 (zero)	Payment at the end of the period
1	Payment at the beginning of the period

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 10 percent, use 10%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 10% for rate and 4 for nper.
- If rate ≤ 0, nper ≤ 0, or pv ≤ 0, CUMIPMT returns the #NUM! error value.
- If start_period < 1, end_period < 1, or start_period > end_period, CUMIPMT returns the #NUM! error value.

- If type is any number other than 0 or 1, CUMIPMT returns the #NUM! error value.

CUMPRINC function

Description

Returns the cumulative principal paid on a loan between `start_period` and `end_period`.

Syntax

```
CUMPRINC(rate, nper, pv, start_period, end_period, type)
```

The CUMPRINC function syntax has the following arguments:

Rate Required. The interest rate.

- **Nper** Required. The total number of payment periods.
- **Pv** Required. The present value.
- **Start_period** Required. The first period in the calculation. Payment periods are numbered beginning with 1.
- **End_period** Required. The last period in the calculation.
- **Type** Required. The timing of the payment.

Type	Timing
0 (zero)	Payment at the end of the period
1	Payment at the beginning of the period

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
- If $rate \leq 0$, $nper \leq 0$, or $pv \leq 0$, CUMPRINC returns the #NUM! error value.

- If $\text{start_period} < 1$, $\text{end_period} < 1$, or $\text{start_period} > \text{end_period}$, CUMPRINC returns the #NUM! error value.
- If type is any number other than 0 or 1, CUMPRINC returns the #NUM! error value.

DB function

Description

Returns the depreciation of an asset for a specified period using the fixed-declining balance method.

Syntax

```
DB(cost, salvage, life, period, [month])
```

The DB function syntax has the following arguments:

- **Cost** Required. The initial cost of the asset.
- **Salvage** Required. The value at the end of the depreciation (sometimes called the salvage value of the asset).
- **Life** Required. The number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).
- **Period** Required. The period for which you want to calculate the depreciation. Period must use the same units as life.
- **Month** Optional. The number of months in the first year. If month is omitted, it is assumed to be 12.

Remarks

- The fixed-declining balance method computes depreciation at a fixed rate. DB uses the following formulas to calculate depreciation for a period:

(cost - total depreciation from prior periods) * rate where:

rate = $1 - ((\text{salvage} / \text{cost}) ^ (1 / \text{life}))$, rounded to three decimal places

- Depreciation for the first and last periods is a special case. For the first period, DB uses this formula:

$\text{cost} * \text{rate} * \text{month} / 12$

- For the last period, DB uses this formula:

$((\text{cost} - \text{total depreciation from prior periods}) * \text{rate} * (12 - \text{month})) / 12$

DDB function

Description

Returns the depreciation of an asset for a specified period using the double-declining balance method or some other method you specify.

Syntax

```
DDB(cost, salvage, life, period, [factor])
```

The DDB function syntax has the following arguments:

Cost Required. The initial cost of the asset.

- **Salvage** Required. The value at the end of the depreciation (sometimes called the salvage value of the asset). This value can be 0.
- **Life** Required. The number of periods over which the asset is being depreciated (sometimes called the useful life of the asset).
- **Period** Required. The period for which you want to calculate the depreciation. Period must use the same units as life.
- **Factor** Optional. The rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method).

IMPORTANT All five arguments must be positive numbers.

Remarks

- The double-declining balance method computes depreciation at an accelerated rate. Depreciation is highest in the first period and decreases in successive periods. DDB uses the following formula to calculate depreciation for a period:

```
Min( (cost - total depreciation from prior periods) * (factor/life), (cost - salvage - total depreciation from prior periods) )
```
- Change factor if you do not want to use the double-declining balance method.

- Use the VDB function if you want to switch to the straight-line depreciation method when depreciation is greater than the declining balance calculation.

DISC function

Description

Returns the discount rate for a security.

Syntax

```
DISC(settlement, maturity, pr, redemption, [basis])
```

The DISC function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Pr** Required. The security's price per \$100 face value.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, 30 years after the January 1, 2008, issue date.

- Settlement, maturity, and basis are truncated to integers.
- If settlement or maturity is not a valid serial date number, DISC returns the #VALUE! error value.
- If $pr \leq 0$ or if redemption ≤ 0 , DISC returns the #NUM! error value.
- If basis < 0 or if basis > 4 , DISC returns the #NUM! error value.
- If settlement \geq maturity, DISC returns the #NUM! error value.
- DISC is calculated as follows:

$$DISC = \frac{\text{redemption} - \text{par}}{\text{par}} \times \frac{B}{DSM}$$

where:

- B = number of days in a year, depending on the year basis.
- DSM = number of days between settlement and maturity.

DOLLARDE function

Description

Converts a dollar price expressed as an integer part and a fraction part, such as 1.02, into a dollar price expressed as a decimal number. Fractional dollar numbers are sometimes used for security prices.

The fraction part of the value is divided by an integer that you specify. For example, if you want your price to be expressed to a precision of 1/16 of a dollar, you divide the fraction part by 16. In this case, 1.02 represents \$1.125 ($\$1 + 2/16 = \1.125).

Syntax

```
DOLLARDE(fractional_dollar, fraction)
```

The DOLLARDE function syntax has the following arguments:

- **Fractional_dollar** Required. A number expressed as an integer part and a fraction part, separated by a decimal symbol.
- **Fraction** Required. The integer to use in the denominator of the fraction.

Remarks

- If fraction is not an integer, it is truncated.
- If fraction is less than 0, DOLLARDE returns the #NUM! error value.
- If fraction is greater than or equal to 0 and less than 1, DOLLARDE returns the #DIV/0! error value.

DOLLARFR function

Description

Converts a dollar price expressed as a decimal number into a dollar price expressed as a fraction. Use DOLLARFR to convert decimal numbers to fractional dollar numbers, such as securities prices.

Syntax

```
DOLLARFR(decimal_dollar, fraction)
```

The DOLLARFR function syntax has the following arguments:

- **Decimal_dollar** Required. A decimal number.
- **Fraction** Required. The integer to use in the denominator of a fraction.

Remarks

- If fraction is not an integer, it is truncated.
- If fraction is less than 0, DOLLARFR returns the #NUM! error value.
- If fraction is 0, DOLLARFR returns the #DIV/0! error value.

DURATION function

Description

Returns the Macauley duration for an assumed par value of \$100. Duration is defined as the weighted average of the present value of the cash flows and is used as a measure of a bond price's response to changes in yield.

Syntax

```
DURATION(settlement, maturity, coupon, yld, frequency, [basis])
```

The DURATION function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Coupon** Required. The security's annual coupon rate.
- **Yld** Required. The security's annual yield.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, frequency, and basis are truncated to integers.
- If settlement or maturity is not a valid date, DURATION returns the #VALUE! error value.
- If coupon < 0 or if yld < 0, DURATION returns the #NUM! error value.
- If frequency is any number other than 1, 2, or 4, DURATION returns the #NUM! error value.
- If basis < 0 or if basis > 4, DURATION returns the #NUM! error value.
- If settlement ≥ maturity, DURATION returns the #NUM! error value.

EFFECT function

Description

Returns the effective annual interest rate, given the nominal annual interest rate and the number of compounding periods per year.

Syntax

```
EFFECT(nominal_rate, npery)
```

The EFFECT function syntax has the following arguments:

- **Nominal_rate** Required. The nominal interest rate.
- **Npery** Required. The number of compounding periods per year.

Remarks

- Npery is truncated to an integer.
- If either argument is nonnumeric, EFFECT returns the #VALUE! error value.
- If nominal_rate ≤ 0 or if npery < 1, EFFECT returns the #NUM! error value.
- EFFECT is calculated as follows:

$$EFFECT = \left(1 + \frac{Nominal_rate}{Npery} \right)^{Npery} - 1$$

FV function

Description

Returns the future value of an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
FV(rate,nper,pmt,[pv],[type])
```

For a more complete description of the arguments in FV and for more information on annuity functions, see PV.

The FV function syntax has the following arguments:

- **Rate** Required. The interest rate per period.
- **Nper** Required. The total number of payment periods in an annuity.
- **Pmt** Required. The payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes. If pmt is omitted, you must include the pv argument.
- **Pv** Optional. The present value, or the lump-sum amount that a series of future payments is worth right now. If pv is omitted, it is assumed to be 0 (zero), and you must include the pmt argument.
- **Type** Optional. The number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

Set type equal to	If payments are due
0	At the end of the period
1	At the beginning of the period

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
- For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.

FVSCCHEDULE function

Description

Returns the future value of an initial principal after applying a series of compound interest rates. Use FVSCCHEDULE to calculate the future value of an investment with a variable or adjustable rate.

Syntax

```
FVSCCHEDULE(principal, schedule)
```

The FVSCCHEDULE function syntax has the following arguments:

- **Principal** Required. The present value.
- **Schedule** Required. An array of interest rates to apply.

Remarks

The values in schedule can be numbers or blank cells; any other value produces the #VALUE! error value for FVSCCHEDULE. Blank cells are taken as zeros (no interest).

INTRATE function

Description

Returns the interest rate for a fully invested security.

Syntax

```
INTRATE(settlement, maturity, investment, redemption, [basis])
```

The INTRATE function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Investment** Required. The amount invested in the security.
- **Redemption** Required. The amount to be received at maturity.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- Settlement, maturity, and basis are truncated to integers.
- If settlement or maturity is not a valid date, INTRATE returns the #VALUE! error value.
- If investment ≤ 0 or if redemption ≤ 0 , INTRATE returns the #NUM! error value.
- If basis < 0 or if basis > 4 , INTRATE returns the #NUM! error value.
- If settlement \geq maturity, INTRATE returns the #NUM! error value.
- INTRATE is calculated as follows:

$$INTRATE = \frac{\text{redemption} - \text{investment}}{\text{investment}} \times \frac{B}{DIM}$$

where:

- B = number of days in a year, depending on the year basis.
- DIM = number of days from settlement to maturity.

IPMT function

Description

Returns the interest payment for a given period for an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
IPMT(rate, per, nper, pv, [fv], [type])
```

The IPMT function syntax has the following arguments:

- **Rate** Required. The interest rate per period.
- **Per** Required. The period for which you want to find the interest and must be in the range 1 to nper.
- **Nper** Required. The total number of payment periods in an annuity.
- **Pv** Required. The present value, or the lump-sum amount that a series of future payments is worth right now.
- **Fv** Optional. The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).
- **Type** Optional. The number 0 or 1 and indicates when payments are due. If type is omitted, it is assumed to be 0.

Set type equal to	If payments are due
0	At the end of the period
1	At the beginning of the period

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate

and 4×12 for $nper$. If you make annual payments on the same loan, use 12% for rate and 4 for $nper$.

- For all the arguments, cash you pay out, such as deposits to savings, is represented by negative numbers; cash you receive, such as dividend checks, is represented by positive numbers.

IRR function

Description

Returns the internal rate of return for a series of cash flows represented by the numbers in values. These cash flows do not have to be even, as they would be for an annuity. However, the cash flows must occur at regular intervals, such as monthly or annually. The internal rate of return is the interest rate received for an investment consisting of payments (negative values) and income (positive values) that occur at regular periods.

Syntax

```
IRR(values, [guess])
```

The IRR function syntax has the following arguments:

- **Values** Required. An array or a reference to cells that contain numbers for which you want to calculate the internal rate of return.
 - Values must contain at least one positive value and one negative value to calculate the internal rate of return.
 - IRR uses the order of values to interpret the order of cash flows. Be sure to enter your payment and income values in the sequence you want.
 - If an array or reference argument contains text, logical values, or empty cells, those values are ignored.
- **Guess** Optional. A number that you guess is close to the result of IRR.
 - In most cases you do not need to provide guess for the IRR calculation. If guess is omitted, it is assumed to be 0.1 (10 percent).
 - If IRR gives the #NUM! error value, or if the result is not close to what you expected, try again with a different value for guess.

Remarks

IRR is closely related to NPV, the net present value function. The rate of return calculated by IRR is the interest rate corresponding to a 0 (zero) net present value. The following formula demonstrates how NPV and IRR are related:

```
NPV ( IRR ( B1 : B6 ) , B1 : B6 )
```

equals 3.60E-08 [Within the accuracy of the IRR calculation, the value 3.60E-08 is effectively 0 (zero).]

ISPMT function

Description

Calculates the interest paid during a specific period of an investment. This function is provided for compatibility with Lotus 1-2-3.

Syntax

```
ISPMT(rate, per, nper, pv)
```

The ISPMT function syntax has the following arguments:

- **Rate** Required. The interest rate for the investment.
- **Per** Required. The period for which you want to find the interest, and must be between 1 and nper.
- **Nper** Required. The total number of payment periods for the investment.
- **Pv** Required. The present value of the investment. For a loan, pv is the loan amount.

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
- For all the arguments, the cash you pay out, such as deposits to savings or other withdrawals, is represented by negative numbers; the cash you receive, such as dividend checks and other deposits, is represented by positive numbers.
- For additional information about financial functions, see the PV function.

MDURATION function

Description

Returns the modified Macauley duration for a security with an assumed par value of \$100.

Syntax

```
MDURATION(settlement, maturity, coupon, yld, frequency, [basis])
```

The MDURATION function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Coupon** Required. The security's annual coupon rate.
- **Yld** Required. The security's annual yield.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date is January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, frequency, and basis are truncated to integers.
- If settlement or maturity is not a valid date, MDURATION returns the #VALUE! error value.
- If yld < 0 or if coupon < 0, MDURATION returns the #NUM! error value.
- If frequency is any number other than 1, 2, or 4, MDURATION returns the #NUM! error value.
- If basis < 0 or if basis > 4, MDURATION returns the #NUM! error value.
- If settlement ≥ maturity, MDURATION returns the #NUM! error value.
- Modified duration is defined as follows:

$$\text{MDURATION} = \frac{\text{DURATION}}{1 + \left(\frac{\text{Market yield}}{\text{Coupon payments per year}} \right)}$$

MIRR function

Description

Returns the modified internal rate of return for a series of periodic cash flows. MIRR considers both the cost of the investment and the interest received on reinvestment of cash.

Syntax

```
MIRR(values, finance_rate, reinvest_rate)
```

The MIRR function syntax has the following arguments:

- **Values** Required. An array or a reference to cells that contain numbers. These numbers represent a series of payments (negative values) and income (positive values) occurring at regular periods.
 - Values must contain at least one positive value and one negative value to calculate the modified internal rate of return. Otherwise, MIRR returns the #DIV/0! error value.
 - If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- **Finance_rate** Required. The interest rate you pay on the money used in the cash flows.
- **Reinvest_rate** Required. The interest rate you receive on the cash flows as you reinvest them.

Remarks

- MIRR uses the order of values to interpret the order of cash flows. Be sure to enter your payment and income values in the sequence you want and with the correct signs (positive values for cash received, negative values for cash paid).
- If n is the number of cash flows in values, f rate is the finance_rate, and r rate is the reinvest_rate, then the formula for MIRR is:

$$\left(\frac{-\text{NPV}(r\text{rate}, \text{values}[\text{positive}]) * (1 + r\text{rate})^n}{\text{NPV}(f\text{rate}, \text{values}[\text{negative}]) * (1 + f\text{rate})} \right)^{\frac{1}{n-1}} - 1$$

NOMINAL function

Description

Returns the nominal annual interest rate, given the effective rate and the number of compounding periods per year.

Syntax

```
NOMINAL(effect_rate, npery)
```

The NOMINAL function syntax has the following arguments:

- **Effect_rate** Required. The effective interest rate.
- **Npery** Required. The number of compounding periods per year.

Remarks

- Npery is truncated to an integer.
- If either argument is nonnumeric, NOMINAL returns the #VALUE! error value.
- If effect_rate ≤ 0 or if npery < 1, NOMINAL returns the #NUM! error value.
- NOMINAL is related to EFFECT as shown in the following equation:

$$EFFECT = \left(1 + \frac{Nominal_rate}{Npery} \right)^{Npery} - 1$$

NPER function

Description

Returns the number of periods for an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
NPER (rate, pmt, pv, [fv], [type])
```

For a more complete description of the arguments in NPER and for more information about annuity functions, see PV.

The NPER function syntax has the following arguments:

- **Rate** Required. The interest rate per period.
- **Pmt** Required. The payment made each period; it cannot change over the life of the annuity. Typically, pmt contains principal and interest but no other fees or taxes.
- **Pv** Required. The present value, or the lump-sum amount that a series of future payments is worth right now.
- **Fv** Optional. The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).
- **Type** Optional. The number 0 or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

NPV function

Description

Calculates the net present value of an investment by using a discount rate and a series of future payments (negative values) and income (positive values).

Syntax

```
NPV(rate,value1,[value2],...)
```

The NPV function syntax has the following arguments:

- **Rate** Required. The rate of discount over the length of one period.
- **Value1, value2, ...** Value1 is required, subsequent values are optional. 1 to 254 arguments representing the payments and income.
 - Value1, value2, ... must be equally spaced in time and occur at the end of each period.
 - NPV uses the order of value1, value2, ... to interpret the order of cash flows. Be sure to enter your payment and income values in the correct sequence.
 - Arguments that are empty cells, logical values, or text representations of numbers, error values, or text that cannot be translated into numbers are ignored.
 - If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored.

Remarks

- The NPV investment begins one period before the date of the value1 cash flow and ends with the last cash flow in the list. The NPV calculation is based on future cash flows. If your first cash flow occurs at the beginning of the first period, the first value must be added to the NPV result, not included in the values arguments. For more information, see the examples below.
- If n is the number of cash flows in the list of values, the formula for NPV is:

$$\text{NPV} = \sum_{j=1}^n \frac{\text{values}_j}{(1 + \text{rate})^j}$$

- NPV is similar to the PV function (present value). The primary difference between PV and NPV is that PV allows cash flows to begin either at the end or at the beginning of the period. Unlike the variable NPV cash flow values, PV cash flows must be constant throughout the investment. For information about annuities and financial functions, see PV.
- NPV is also related to the IRR function (internal rate of return). IRR is the rate for which NPV equals zero: $\text{NPV}(\text{IRR}(\dots), \dots) = 0$.

ODDFPRICE function

Description

Returns the price per \$100 face value of a security having an odd (short or long) first period.

Syntax

```
ODDFPRICE(settlement, maturity, issue, first_coupon, rate, yld, redemption, frequency, [basis])
```

The ODDFPRICE function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Issue** Required. The security's issue date.
- **First_coupon** Required. The security's first coupon date.
- **Rate** Required. The security's interest rate.
- **Yld** Required. The security's annual yield.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, issue, first_coupon, and basis are truncated to integers.
- If settlement, maturity, issue, or first_coupon is not a valid date, ODDFPRICE returns the #VALUE! error value.
- If rate < 0 or if yld < 0, ODDFPRICE returns the #NUM! error value.
- If basis < 0 or if basis > 4, ODDFPRICE returns the #NUM! error value.
- The following date condition must be satisfied; otherwise, ODDFPRICE returns the #NUM! error value:

$$\text{maturity} > \text{first_coupon} > \text{settlement} > \text{issue}$$

- ODDFPRICE is calculated as follows:

Odd short first coupon:

$$\begin{aligned} \text{ODDFPRICE} = & \left[\frac{\text{redemption}}{\left(1 + \frac{\text{yld}}{\text{frequency}}\right)^{\left(N + \frac{\text{ISC}}{E}\right)}} \right] + \left[\frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{\text{DFC}}{E}}{\left(1 + \frac{\text{yld}}{\text{frequency}}\right)^{\frac{\text{ISC}}{E}}} \right] \\ & + \left[\sum_{k=2}^N \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{\left(1 + \frac{\text{yld}}{\text{frequency}}\right)^{\left(k + \frac{\text{ISC}}{E}\right)}} \right] \\ & - \left[100 \times \frac{\text{rate}}{\text{frequency}} \times \frac{A}{E} \right] \end{aligned}$$

where:

- A = number of days from the beginning of the coupon period to the settlement date (accrued days).
- DSC = number of days from the settlement to the next coupon date.
- DFC = number of days from the beginning of the odd first coupon to the first coupon date.
- E = number of days in the coupon period.
- N = number of coupons payable between the settlement date and the redemption date. (If this number contains a fraction, it is raised to the next whole number.)

Odd long first coupon:

$$\begin{aligned}
 ODDPRICE = & \left[\frac{\text{redemption}}{\left(1 + \frac{yld}{\text{frequency}}\right)^{\left(N + N_0 + \frac{DSC}{E}\right)}} \right] \\
 & + \left[\frac{100 \times \frac{\text{rate}}{\text{frequency}} \times \left[\sum_{j=1}^{NC} \frac{DC_j}{NL_j} \right]}{\left(1 + \frac{yld}{\text{frequency}}\right)^{\left(N_0 + \frac{DSC}{E}\right)}} \right] \\
 & + \left[\sum_{k=1}^N \frac{100 \times \frac{\text{rate}}{\text{frequency}}}{\left(1 + \frac{yld}{\text{frequency}}\right)^{\left(k - N_0 + \frac{DSC}{E}\right)}} \right] \\
 & - \left[100 \times \frac{\text{rate}}{\text{frequency}} \times \sum_{j=1}^{NC} \frac{A_j}{NL_j} \right]
 \end{aligned}$$

where:

- Ai = number of days from the beginning of the ith, or last, quasi-coupon period within odd period.
- DCi = number of days from dated date (or issue date) to first quasi-coupon (i = 1) or number of days in quasi-coupon (i = 2, ..., i = NC).

- DSC = number of days from settlement to next coupon date.
- E = number of days in coupon period.
- N = number of coupons payable between the first real coupon date and redemption date. (If this number contains a fraction, it is raised to the next whole number.)
- NC = number of quasi-coupon periods that fit in odd period. (If this number contains a fraction, it is raised to the next whole number.)
- NLi = normal length in days of the full i th, or last, quasi-coupon period within odd period.
- Nq = number of whole quasi-coupon periods between settlement date and first coupon.

ODDFYIELD function

Description

Returns the yield of a security that has an odd (short or long) first period.

Syntax

```
ODDFYIELD(settlement, maturity, issue, first_coupon, rate, pr, redemption, frequency, [basis])
```

The ODDFYIELD function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Issue** Required. The security's issue date.
- **First_coupon** Required. The security's first coupon date.
- **Rate** Required. The security's interest rate.
- **Pr** Required. The security's price.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360

3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, issue, first_coupon, and basis are truncated to integers.
- If settlement, maturity, issue, or first_coupon is not a valid date, ODDFYIELD returns the #VALUE! error value.
- If rate < 0 or if pr ≤ 0, ODDFYIELD returns the #NUM! error value.
- If basis < 0 or if basis > 4, ODDFYIELD returns the #NUM! error value.
- The following date condition must be satisfied; otherwise, ODDFYIELD returns the #NUM! error value:

maturity > first_coupon > settlement > issue

ODDLPRICE function

Description

Returns the price per \$100 face value of a security having an odd (short or long) last coupon period.

Syntax

```
ODDLPRICE(settlement, maturity, last_interest, rate, yld, redemption, frequency, [basis])
```

The ODDLPRICE function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Last_interest** Required. The security's last coupon date.
- **Rate** Required. The security's interest rate.
- **Yld** Required. The security's annual yield.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, last_interest, and basis are truncated to integers.
- If settlement, maturity, or last_interest is not a valid date, ODDLPRICE returns the #VALUE! error value.
- If rate < 0 or if yld < 0, ODDLPRICE returns the #NUM! error value.
- If basis < 0 or if basis > 4, ODDLPRICE returns the #NUM! error value.
- The following date condition must be satisfied; otherwise, ODDLPRICE returns the #NUM! error value:

maturity > settlement > last_interest

ODDLYIELD function

Description

Returns the yield of a security that has an odd (short or long) last period.

Syntax

```
ODDLYIELD(settlement, maturity, last_interest, rate, pr, redemption, frequency, [basis])
```

The ODDLYIELD function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Last_interest** Required. The security's last coupon date.
- **Rate** Required. The security's interest rate
- **Pr** Required. The security's price.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, last_interest, and basis are truncated to integers.
- If settlement, maturity, or last_interest is not a valid date, ODDLYIELD returns the #VALUE! error value.
- If rate < 0 or if pr ≤ 0, ODDLYIELD returns the #NUM! error value.
- If basis < 0 or if basis > 4, ODDLYIELD returns the #NUM! error value.
- The following date condition must be satisfied; otherwise, ODDLYIELD returns the #NUM! error value:

$$\text{maturity} > \text{settlement} > \text{last_interest}$$

- ODDLYIELD is calculated as follows:

$$\text{ODDLYIELD} = \frac{\left[\text{redemption} + \left(\left(\sum_{j=1}^{MC} \frac{DC_j}{NL_j} \right) \times \frac{100 \times \text{rate}}{\text{frequency}} \right) \right] - \left[\text{par} + \left(\left(\sum_{j=1}^{MC} \frac{A_j}{NL_j} \right) \times \frac{100 \times \text{rate}}{\text{frequency}} \right) \right]}{\left[\text{par} + \left(\left(\sum_{j=1}^{MC} \frac{A_j}{NL_j} \right) \times \frac{100 \times \text{rate}}{\text{frequency}} \right) \right]} \times \frac{\text{frequency}}{\left[\left(\sum_{j=1}^{MC} \frac{DSC_j}{NL_j} \right) \right]}$$

where:

- A_i = number of accrued days for the i th, or last, quasi-coupon period within odd period counting forward from last interest date before redemption.

- DC_i = number of days counted in the i th, or last, quasi-coupon period as delimited by the length of the actual coupon period.
- NC = number of quasi-coupon periods that fit in odd period; if this number contains a fraction it will be raised to the next whole number.
- NL_i = normal length in days of the i th, or last, quasi-coupon period within odd coupon period.

PMT function

Description

Calculates the payment for a loan based on constant payments and a constant interest rate.

Syntax

```
PMT(rate, nper, pv, [fv], [type])
```

NOTE For a more complete description of the arguments in PMT, see the PV function.

The PMT function syntax has the following arguments:

- **Rate** Required. The interest rate for the loan.
- **Nper** Required. The total number of payments for the loan.
- **Pv** Required. The present value, or the total amount that a series of future payments is worth now; also known as the principal.
- **Fv** Optional. The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (zero), that is, the future value of a loan is 0.
- **Type** Optional. The number 0 (zero) or 1 and indicates when payments are due.

Set type equal to	If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

Remarks

- The payment returned by PMT includes principal and interest but no taxes, reserve payments, or fees sometimes associated with loans.
- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at an annual interest rate of 12 percent, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12 percent for rate and 4 for nper.

PPMT function

Description

Returns the payment on the principal for a given period for an investment based on periodic, constant payments and a constant interest rate.

Syntax

```
PPMT(rate, per, nper, pv, [fv], [type])
```

NOTE For a more complete description of the arguments in PPMT, see PV.

The PPMT function syntax has the following arguments:

- **Rate** Required. The interest rate per period.
- **Per** Required. Specifies the period and must be in the range 1 to nper.
- **Nper** Required. The total number of payment periods in an annuity.
- **Pv** Required. The present value — the total amount that a series of future payments is worth now.
- **Fv** Optional. The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (zero), that is, the future value of a loan is 0.
- **Type** Optional. The number 0 or 1 and indicates when payments are due.

• Set type equal to

0 or omitted

1

• If payments are due

At the end of the period

At the beginning of the period

Remarks

Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.

PRICE function

Description

Returns the price per \$100 face value of a security that pays periodic interest.

Syntax

```
PRICE(settlement, maturity, rate, yld, redemption, frequency, [basis])
```

IMPORTANT Dates should be entered by using the DATE function, or as results of other formulas or functions. For example, use DATE(2008,5,23) for the 23rd day of May, 2008. Problems can occur if dates are entered as text.

The PRICE function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Rate** Required. The security's annual coupon rate.
- **Yld** Required. The security's annual yield.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, frequency, and basis are truncated to integers.
- If settlement or maturity is not a valid date, PRICE returns the #VALUE! error value.
- If $yld < 0$ or if $rate < 0$, PRICE returns the #NUM! error value.
- If $redemption \leq 0$, PRICE returns the #NUM! error value.
- If frequency is any number other than 1, 2, or 4, PRICE returns the #NUM! error value.
- If $basis < 0$ or if $basis > 4$, PRICE returns the #NUM! error value.
- If $settlement \geq maturity$, PRICE returns the #NUM! error value.
- PRICE is calculated as follows:

$$PRICE = \left[\frac{redemption}{\left(1 + \frac{yld}{frequency}\right)^{\left(N-1 + \frac{DSC}{E}\right)}} \right] + \left[\sum_{k=1}^N \frac{100 \times \frac{rate}{frequency}}{\left(1 + \frac{yld}{frequency}\right)^{\left(k-1 + \frac{DSC}{E}\right)}} \right] - \left(100 \times \frac{rate}{frequency} \times \frac{A}{E} \right)$$

where:

- DSC = number of days from settlement to next coupon date.
- E = number of days in coupon period in which the settlement date falls.
- N = number of coupons payable between settlement date and redemption date.
- A = number of days from beginning of coupon period to settlement date.

PRICEDISC function

Description

Returns the price per \$100 face value of a discounted security.

Syntax

```
PRICEDISC(settlement, maturity, discount, redemption, [basis])
```

The PRICEDISC function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Discount** Required. The security's discount rate.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the

settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- Settlement, maturity, and basis are truncated to integers.
- If settlement or maturity is not a valid date, PRICEDISC returns the #VALUE! error value.
- If discount ≤ 0 or if redemption ≤ 0 , PRICEDISC returns the #NUM! error value.
- If basis < 0 or if basis > 4 , PRICEDISC returns the #NUM! error value.
- If settlement \geq maturity, PRICEDISC returns the #NUM! error value.
- PRICEDISC is calculated as follows:

$$PRICEDISC = redemption - discount \times redemption \times \frac{DSM}{B}$$

where:

- B = number of days in year, depending on year basis.
- DSM = number of days from settlement to maturity.

PRICEMAT function



Description

Returns the price per \$100 face value of a security that pays interest at maturity.

Syntax

```
PRICEMAT(settlement, maturity, issue, rate, yld, [basis])
```

The PRICEMAT function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Issue** Required. The security's issue date, expressed as a serial date number.
- **Rate** Required. The security's interest rate at date of issue.
- **Yld** Required. The security's annual yield.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 (zero) or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- Settlement, maturity, issue, and basis are truncated to integers.
- If settlement, maturity, or issue is not a valid date, PRICEMAT returns the #VALUE! error value.
- If rate < 0 or if yld < 0, PRICEMAT returns the #NUM! error value.
- If basis < 0 or if basis > 4, PRICEMAT returns the #NUM! error value.
- If settlement ≥ maturity, PRICEMAT returns the #NUM! error value.
- PRICEMAT is calculated as follows:

$$PRICEMAT = \frac{100 + \left(\frac{DIM}{B} \times rate \times 100\right)}{1 + \left(\frac{DSM}{B} \times yld\right)} - \left(\frac{A}{B} \times rate \times 100\right)$$

where:

- B = number of days in year, depending on year basis.
- DSM = number of days from settlement to maturity.
- DIM = number of days from issue to maturity.
- A = number of days from issue to settlement.

PV function

Description

Returns the present value of an investment. The present value is the total amount that a series of future payments is worth now. For example, when you borrow money, the loan amount is the present value to the lender.

Syntax

```
PV(rate, nper, pmt, [fv], [type])
```

The PV function syntax has the following arguments:

- **Rate** Required. The interest rate per period. For example, if you obtain an automobile loan at a 10 percent annual interest rate and make monthly payments, your interest rate per month is 10%/12, or 0.83%. You would enter 10%/12, or 0.83%, or 0.0083, into the formula as the rate.
- **Nper** Required. The total number of payment periods in an annuity. For example, if you get a four-year car loan and make monthly payments, your loan has 4*12 (or 48) periods. You would enter 48 into the formula for nper.
- **Pmt** Required. The payment made each period and cannot change over the life of the annuity. Typically, pmt includes principal and interest but no other fees or taxes. For example, the monthly payments on a \$10,000, four-year car loan at 12 percent are \$263.33. You would enter -263.33 into the formula as the pmt. If pmt is omitted, you must include the fv argument.
- **Fv** Optional. The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0). For example, if you want to save \$50,000 to pay for a special project in 18 years, then \$50,000 is the future value. You could then make a conservative guess at an interest rate and determine how much you must save each month. If fv is omitted, you must include the pmt argument.
- **Type** Optional. The number 0 or 1 and indicates when payments are due.

Set type equal to

If payments are due

0 or omitted

At the end of the period

Remarks

- Make sure that you are consistent about the units you use for specifying rate and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for rate and 4*12 for nper. If you make annual payments on the same loan, use 12% for rate and 4 for nper.
- The following functions apply to annuities:

CUMIPMT	PPMT
---------	------

CUMPRINC	PV
----------	----

FV	RATE
----	------

FVSCHEDULE	XIRR
------------	------

IPMT	XNPV
------	------

PMT

- An annuity is a series of constant cash payments made over a continuous period. For example, a car loan or a mortgage is an annuity. For more information, see the description for each annuity function.
- In annuity functions, cash you pay out, such as a deposit to savings, is represented by a negative number; cash you receive, such as a dividend check, is represented by a positive number. For example, a \$1,000 deposit to the bank would be represented by the argument -1000 if you are the depositor and by the argument 1000 if you are the bank.
- Solves for one financial argument in terms of the others. If rate is not 0, then:

$$pv * (1 + rate)^{nper} + pmt(1 + rate * type) * \left(\frac{(1 + rate)^{nper} - 1}{rate} \right) + fv = 0$$

If rate is 0, then:

$$(pmt * nper) + pv + fv = 0$$

RATE function

Description

Returns the interest rate per period of an annuity. RATE is calculated by iteration and can have zero or more solutions. If the successive results of RATE do not converge to within 0.0000001 after 20 iterations, RATE returns the #NUM! error value.

Syntax

```
RATE(nper, pmt, pv, [fv], [type], [guess])
```

NOTE For a complete description of the arguments nper, pmt, pv, fv, and type, see PV.

The RATE function syntax has the following arguments:

- **Nper** Required. The total number of payment periods in an annuity.
- **Pmt** Required. The payment made each period and cannot change over the life of the annuity. Typically, pmt includes principal and interest but no other fees or taxes. If pmt is omitted, you must include the fv argument.
- **Pv** Required. The present value — the total amount that a series of future payments is worth now.
- **Fv** Optional. The future value, or a cash balance you want to attain after the last payment is made. If fv is omitted, it is assumed to be 0 (the future value of a loan, for example, is 0).
- **Type** Optional. The number 0 or 1 and indicates when payments are due.

• Set type equal to	• If payments are due
0 or omitted	At the end of the period
1	At the beginning of the period

- **Guess** Optional. Your guess for what the rate will be.
 - If you omit guess, it is assumed to be 10 percent.
 - If RATE does not converge, try different values for guess. RATE usually converges if guess is between 0 and 1.

Remarks

Make sure that you are consistent about the units you use for specifying guess and nper. If you make monthly payments on a four-year loan at 12 percent annual interest, use 12%/12 for guess and 4*12 for nper. If you make annual payments on the same loan, use 12% for guess and 4 for nper.

RECEIVED function

Description

Returns the amount received at maturity for a fully invested security.

Syntax

```
RECEIVED(settlement, maturity, investment, discount, [basis])
```

The RECEIVED function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Investment** Required. The amount invested in the security.
- **Discount** Required. The security's discount rate.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the

settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- Settlement, maturity, and basis are truncated to integers.
- If settlement or maturity is not a valid date, RECEIVED returns the #VALUE! error value.
- If investment ≤ 0 or if discount ≤ 0 , RECEIVED returns the #NUM! error value.
- If basis < 0 or if basis > 4 , RECEIVED returns the #NUM! error value.
- If settlement \geq maturity, RECEIVED returns the #NUM! error value.
- RECEIVED is calculated as follows:

$$RECEIVED = \frac{\textit{investment}}{1 - (\textit{discount} \times \frac{\textit{DIM}}{\textit{B}})}$$

where:

- B = number of days in a year, depending on the year basis.
- DIM = number of days from issue to maturity.

SLN function



Description

Returns the straight-line depreciation of an asset for one period.

Syntax

```
SLN(cost, salvage, life)
```

The SLN function syntax has the following arguments:

- **Cost** Required. The initial cost of the asset.
- **Salvage** Required. The value at the end of the depreciation (sometimes called the salvage value of the asset).
- **Life** Required. The number of periods over which the asset is depreciated (sometimes called the useful life of the asset).

SYD function

Description

Returns the sum-of-years' digits depreciation of an asset for a specified period.

Syntax

```
SYD(cost, salvage, life, per)
```

The SYD function syntax has the following arguments:

- **Cost** Required. The initial cost of the asset.
- **Salvage** Required. The value at the end of the depreciation (sometimes called the salvage value of the asset).
- **Life** Required. The number of periods over which the asset is depreciated (sometimes called the useful life of the asset).
- **Per** Required. The period and must use the same units as life.

Remarks

SYD is calculated as follows:

$$SYD = \frac{(cost - salvage) * (life - per + 1) * 2}{(life)(life + 1)}$$

TBILLEQ function



Description

Returns the bond-equivalent yield for a Treasury bill.

Syntax

```
TBILLEQ(settlement, maturity, discount)
```

The TBILLEQ function syntax has the following arguments:

Settlement Required. The Treasury bill's settlement date. The security settlement date is the date after the issue date when the Treasury bill is traded to the buyer.

- **Maturity** Required. The Treasury bill's maturity date. The maturity date is the date when the Treasury bill expires.
- **Discount** Required. The Treasury bill's discount rate.

Remarks

- Settlement and maturity are truncated to integers.
- If settlement or maturity is not a valid date, TBILLEQ returns the #VALUE! error value.
- If discount ≤ 0 , TBILLEQ returns the #NUM! error value.
- If settlement > maturity, or if maturity is more than one year after settlement, TBILLEQ returns the #NUM! error value.
- TBILLEQ is calculated as $TBILLEQ = (365 \times \text{rate}) / (360 - (\text{rate} \times \text{DSM}))$, where DSM is the number of days between settlement and maturity computed according to the 360 days per year basis.

TBILLPRICE function



Description

Returns the price per \$100 face value for a Treasury bill.

Syntax

```
TBILLPRICE(settlement, maturity, discount)
```

The TBILLPRICE function syntax has the following arguments:

- **Settlement** Required. The Treasury bill's settlement date. The security settlement date is the date after the issue date when the Treasury bill is traded to the buyer.
- **Maturity** Required. The Treasury bill's maturity date. The maturity date is the date when the Treasury bill expires.
- **Discount** Required. The Treasury bill's discount rate.

Remarks

- Settlement and maturity are truncated to integers.
- If settlement or maturity is not a valid date, TBILLPRICE returns the #VALUE! error value.
- If discount ≤ 0 , TBILLPRICE returns the #NUM! error value.
- If settlement > maturity, or if maturity is more than one year after settlement, TBILLPRICE returns the #NUM! error value.
- TBILLPRICE is calculated as follows:

$$TBILLPRICE = 100 \times \left(1 - \frac{\text{discount} \times DSM}{360}\right)$$

where:

- DSM = number of days from settlement to maturity, excluding any maturity date that is more than one calendar year after the settlement date.

TBILLYIELD function



Description

Returns the yield for a Treasury bill.

Syntax

```
TBILLYIELD(settlement, maturity, pr)
```

The TBILLYIELD function syntax has the following arguments:

- **Settlement** Required. The Treasury bill's settlement date. The security settlement date is the date after the issue date when the Treasury bill is traded to the buyer.
- **Maturity** Required. The Treasury bill's maturity date. The maturity date is the date when the Treasury bill expires.
- **Pr** Required. The Treasury bill's price per \$100 face value.

Remarks

- Settlement and maturity are truncated to integers.
- If settlement or maturity is not a valid date, TBILLYIELD returns the #VALUE! error value.
- If $pr \leq 0$, TBILLYIELD returns the #NUM! error value.
- If $settlement \geq maturity$, or if maturity is more than one year after settlement, TBILLYIELD returns the #NUM! error value.
- TBILLYIELD is calculated as follows:

$$TBILLYIELD = \frac{100 - pr}{pr} \times \frac{360}{DSM}$$

where:

- DSM = number of days from settlement to maturity, excluding any maturity date that is more than one calendar year after the settlement date.

VDB function



Description

Returns the depreciation of an asset for any period you specify, including partial periods, using the double-declining balance method or some other method you specify. VDB stands for variable declining balance.

Syntax

```
VDB(cost, salvage, life, start_period, end_period, [factor], [no_switch])
```

The VDB function syntax has the following arguments:

- **Cost** Required. The initial cost of the asset.
- **Salvage** Required. The value at the end of the depreciation (sometimes called the salvage value of the asset). This value can be 0.
- **Life** Required. The number of periods over which the asset is depreciated (sometimes called the useful life of the asset).
- **Start_period** Required. The starting period for which you want to calculate the depreciation. Start_period must use the same units as life.
- **End_period** Required. The ending period for which you want to calculate the depreciation. End_period must use the same units as life.
- **Factor** Optional. The rate at which the balance declines. If factor is omitted, it is assumed to be 2 (the double-declining balance method). Change factor if you do not want to use the double-declining balance method. For a description of the double-declining balance method, see DDB.
- **No_switch** Optional. A logical value specifying whether to switch to straight-line depreciation when depreciation is greater than the declining balance calculation.
 - If no_switch is TRUE, does not switch to straight-line depreciation even when the depreciation is greater than the declining balance calculation.
 - If no_switch is FALSE or omitted, switches to straight-line depreciation when depreciation is greater than the declining balance calculation.

IMPORTANT All arguments except no_switch must be positive numbers.

XIRR function



Description

Returns the internal rate of return for a schedule of cash flows that is not necessarily periodic. To calculate the internal rate of return for a series of periodic cash flows, use the IRR function.

Syntax

```
XIRR(values, dates, [guess])
```

The XIRR function syntax has the following arguments:

- **Values** Required. A series of cash flows that corresponds to a schedule of payments in dates. The first payment is optional and corresponds to a cost or payment that occurs at the beginning of the investment. If the first value is a cost or payment, it must be a negative value. All succeeding payments are discounted based on a 365-day year. The series of values must contain at least one positive and one negative value.
- **Dates** Required. A schedule of payment dates that corresponds to the cash flow payments. Dates may occur in any order. Dates should be entered by using the DATE function, or as results of other formulas or functions. For example, use DATE(2008,5,23) for the 23rd day of May, 2008. Problems can occur if dates are entered as text. .
- **Guess** Optional. A number that you guess is close to the result of XIRR.

Remarks

- Numbers in dates are truncated to integers.
- XIRR expects at least one positive cash flow and one negative cash flow; otherwise, XIRR returns the #NUM! error value.
- If any number in dates is not a valid date, XIRR returns the #VALUE! error value.
- If any number in dates precedes the starting date, XIRR returns the #NUM! error value.
- If values and dates contain a different number of values, XIRR returns the #NUM! error value.

- In most cases you do not need to provide guess for the XIRR calculation. If omitted, guess is assumed to be 0.1 (10 percent).
- XIRR is closely related to XNPV, the net present value function. The rate of return calculated by XIRR is the interest rate corresponding to XNPV = 0.
- Uses an iterative technique for calculating XIRR. Using a changing rate (starting with guess), XIRR cycles through the calculation until the result is accurate within 0.000001 percent. If XIRR can't find a result that works after 100 tries, the #NUM! error value is returned. The rate is changed until:

$$0 = \sum_{i=1}^N \frac{P_i}{(1 + rate)^{\frac{(d_i - d_1)}{365}}}$$

where:

- d_i = the i th, or last, payment date.
- d_1 = the 0th payment date.
- P_i = the i th, or last, payment.

XNPV function



Description

Returns the net present value for a schedule of cash flows that is not necessarily periodic. To calculate the net present value for a series of cash flows that is periodic, use the NPV function.

Syntax

```
XNPV(rate, values, dates)
```

The XNPV function syntax has the following arguments:

- **Rate** Required. The discount rate to apply to the cash flows.
- **Values** Required. A series of cash flows that corresponds to a schedule of payments in dates. The first payment is optional and corresponds to a cost or payment that occurs at the beginning of the investment. If the first value is a cost or payment, it must be a negative value. All succeeding payments are discounted based on a 365-day year. The series of values must contain at least one positive value and one negative value.
- **Dates** Required. A schedule of payment dates that corresponds to the cash flow payments. The first payment date indicates the beginning of the schedule of payments. All other dates must be later than this date, but they may occur in any order.

Remarks

- Numbers in dates are truncated to integers.
- If any argument is nonnumeric, XNPV returns the #VALUE! error value.
- If any number in dates is not a valid date, XNPV returns the #VALUE! error value.
- If any number in dates precedes the starting date, XNPV returns the #NUM! error value.
- If values and dates contain a different number of values, XNPV returns the #NUM! error value.
- XNPV is calculated as follows:

$$XNPV = \sum_{i=1}^N \frac{P_i}{(1 + rate)^{\frac{(d_i - d_1)}{365}}}$$

where:

- d_i = the i th, or last, payment date.
- d_1 = the 0th payment date.
- P_i = the i th, or last, payment.

YIELD function



Description

Returns the yield on a security that pays periodic interest. Use YIELD to calculate bond yield.

Syntax

```
YIELD(settlement, maturity, rate, pr, redemption, frequency, [basis])
```

The YIELD function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Rate** Required. The security's annual coupon rate.
- **Pr** Required. The security's price per \$100 face value.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Frequency** Required. The number of coupon payments per year. For annual payments, frequency = 1; for semiannual, frequency = 2; for quarterly, frequency = 4.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360


Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.
- Settlement, maturity, frequency, and basis are truncated to integers.
- If settlement or maturity is not a valid date, YIELD returns the #VALUE! error value.
- If rate < 0, YIELD returns the #NUM! error value.
- If pr ≤ 0 or if redemption ≤ 0, YIELD returns the #NUM! error value.
- If frequency is any number other than 1, 2, or 4, YIELD returns the #NUM! error value.
- If basis < 0 or if basis > 4, YIELD returns the #NUM! error value.
- If settlement ≥ maturity, YIELD returns the #NUM! error value.
- If there is one coupon period or less until redemption, YIELD is calculated as follows:

$$YIELD = \frac{\left(\frac{redemption}{100} + \frac{rate}{frequency}\right) - \left(\frac{par}{100} + \left(\frac{A}{E} \times \frac{rate}{frequency}\right)\right)}{\frac{par}{100} + \left(\frac{A}{E} \times \frac{rate}{frequency}\right)} \times \frac{frequency \times E}{DSR}$$

where:

- A = number of days from the beginning of the coupon period to the settlement date (accrued days).
- DSR = number of days from the settlement date to the redemption date.
- E = number of days in the coupon period.
- If there is more than one coupon period until redemption, YIELD is calculated through a hundred iterations. The resolution uses the Newton method, based on the formula used for the function PRICE. The yield is changed until the estimated price given the yield is close to price.

NOTE To view the number as a percentage, select the cell, and then on the **Home** tab, in the **Number** group, click **Percent Style** .

YIELDDISC function



Description

Returns the annual yield for a discounted security.

Syntax

```
YIELDDISC(settlement, maturity, pr, redemption, [basis])
```

The YIELDDISC function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Pr** Required. The security's price per \$100 face value.
- **Redemption** Required. The security's redemption value per \$100 face value.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1, 2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the

settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- Settlement, maturity, and basis are truncated to integers.
- If settlement or maturity is not a valid date, YIELDDISC returns the #VALUE! error value.
- If $pr \leq 0$ or if redemption ≤ 0 , YIELDDISC returns the #NUM! error value.
- If basis < 0 or if basis > 4 , YIELDDISC returns the #NUM! error value.
- If settlement \geq maturity, YIELDDISC returns the #NUM! error value.

YIELDMAT function



Description

Returns the annual yield of a security that pays interest at maturity.

Syntax

```
YIELDMAT(settlement, maturity, issue, rate, pr, [basis])
```

The YIELDMAT function syntax has the following arguments:

- **Settlement** Required. The security's settlement date. The security settlement date is the date after the issue date when the security is traded to the buyer.
- **Maturity** Required. The security's maturity date. The maturity date is the date when the security expires.
- **Issue** Required. The security's issue date, expressed as a serial date number.
- **Rate** Required. The security's interest rate at date of issue.
- **Pr** Required. The security's price per \$100 face value.
- **Basis** Optional. The type of day count basis to use.

Basis	Day count basis
0 or omitted	US (NASD) 30/360
1	Actual/actual
2	Actual/360
3	Actual/365
4	European 30/360

Remarks

- The settlement date is the date a buyer purchases a coupon, such as a bond. The maturity date is the date when a coupon expires. For example, suppose a 30-year bond is issued on January 1,

2008, and is purchased by a buyer six months later. The issue date would be January 1, 2008, the settlement date would be July 1, 2008, and the maturity date would be January 1, 2038, which is 30 years after the January 1, 2008, issue date.

- Settlement, maturity, issue, and basis are truncated to integers.
- If settlement, maturity, or issue is not a valid date, YIELDMAT returns the #VALUE! error value.
- If rate < 0 or if pr ≤ 0, YIELDMAT returns the #NUM! error value.
- If basis < 0 or if basis > 4, YIELDMAT returns the #NUM! error value.
- If settlement ≥ maturity, YIELDMAT returns the #NUM! error value.

ABS function

Description

Returns the absolute value of a number. The absolute value of a number is the number without its sign.

Syntax

```
ABS (number)
```

The ABS function syntax has the following arguments:

- **Number** Required. The real number of which you want the absolute value.

ACOS function

Description

Returns the arccosine, or inverse cosine, of a number. The arccosine is the angle whose cosine is *number*. The returned angle is given in radians in the range 0 (zero) to pi.

Syntax

```
ACOS (number)
```

The ACOS function syntax has the following arguments:

- **Number** Required. The cosine of the angle you want and must be from -1 to 1.

Remark

If you want to convert the result from radians to degrees, multiply it by 180/PI() or use the DEGREES function.

ACOSH function

Description

Returns the inverse hyperbolic cosine of a number. Number must be greater than or equal to 1. The inverse hyperbolic cosine is the value whose hyperbolic cosine is *number*, so $\text{ACOSH}(\text{COSH}(\text{number}))$ equals *number*.

Syntax

```
ACOSH (number)
```

The ACOSH function syntax has the following arguments:

- **Number** Required. Any real number equal to or greater than 1.

AGGREGATE function

Returns an aggregate in a list or database.

The AGGREGATE function addresses the limitation of conditional formatting. Data bars, Icon Sets and Color Scales cannot display conditional formatting if there are errors in the range. This is because the MIN, MAX and PERCENTILE functions do not calculate when there is an error in the calculation range. The LARGE, SMALL, and STDEVP functions also affect the appropriate functionality of certain conditional formatting rules for the same reasons. By using the AGGREGATE function, you can implement those functions because the errors will be ignored. In addition, the AGGREGATE function can apply different aggregate functions to a list or database with the option to ignore hidden rows and error values.

Syntax

REFERENCE FORM

```
AGGREGATE(function_num, options, ref1, [ref2], ...)
```

ARRAY FORM

```
AGGREGATE(function_num, options, array, [k])
```

The AGGREGATE function syntax has the following arguments:

- **Function_num** Required. A number 1 to 19 that specifies which function to use.

Function_num	Function
1	AVERAGE
2	COUNT
3	COUNTA
4	MAX
5	MIN
6	PRODUCT
7	STDEV.S

8	STDEV.P
9	SUM
10	VAR.S
11	VAR.P
12	MEDIAN
13	MODE.SNGL
14	LARGE
15	SMALL
16	PERCENTILE.INC
17	QUARTILE.INC
18	PERCENTILE.EXC
19	QUARTILE.EXC

- **Options** Required. A numerical value that determines which values to ignore in the evaluation range for the function.

Option	Behavior
0 or omitted	Ignore nested SUBTOTAL and AGGREGATE functions
1	Ignore hidden rows, nested SUBTOTAL and AGGREGATE functions
2	Ignore error values, nested SUBTOTAL and AGGREGATE functions
3	Ignore hidden rows, error values, nested SUBTOTAL and AGGREGATE functions
4	Ignore nothing
5	Ignore hidden rows
6	Ignore error values
7	Ignore hidden rows and error values

- **Ref1** Required. The first numeric argument for functions that take multiple numeric arguments for which you want the aggregate value.
- **Ref2,...** Optional. Numeric arguments 2 to 253 for which you want the aggregate value.

For functions that take an array, ref1 is an array, an array formula, or a reference to a range of cells for which you want the aggregate value. Ref2 is a second argument that is required for certain functions. The following functions require a ref2 argument:

Function

LARGE(array,k)

SMALL(array,k)

PERCENTILE.INC(array,k)

QUARTILE.INC(array,quart)

PERCENTILE.EXC(array,k)

QUARTILE.EXC(array,quart)

Remarks

Function_num:

- As soon as you type the function_num argument when you enter the AGGREGATE function into a cell on the worksheet, you will see a list of all functions that you can use as arguments.

Errors:

- If a second ref argument is required but not provided, AGGREGATE returns a #VALUE! error.
- If one or more of the references are 3-D references, AGGREGATE returns the #VALUE! error value.

Nested Aggregates:

- If there are other AGGREGATEs within ref1, ref2,... (or nested AGGREGATEs), these nested AGGREGATEs are ignored to avoid double counting.
- If there are SUBTOTALs within refs of the AGGREGATE function, these SUBTOTALs are ignored.
- If there are AGGREGATEs within the SUBTOTAL function, these AGGREGATEs are ignored.

Type of Range:

- The AGGREGATE function is designed for columns of data, or vertical ranges. It is not designed for rows of data, or horizontal ranges. For example, when you subtotal a horizontal range using option 1, such as AGGREGATE(1, 1, ref1), hiding a column does not affect the aggregate sum value. But, hiding a row in vertical range does affect the aggregate.

ASIN function

Description

Returns the arcsine, or inverse sine, of a number. The arcsine is the angle whose sine is *number*. The returned angle is given in radians in the range $-\pi/2$ to $\pi/2$.

Syntax

```
ASIN (number)
```

The ASIN function syntax has the following arguments:

- **Number** Required. The sine of the angle you want and must be from -1 to 1.

Remark

To express the arcsine in degrees, multiply the result by $180/\text{PI}()$ or use the DEGREES function.

ASINH function

Description

Returns the inverse hyperbolic sine of a number. The inverse hyperbolic sine is the value whose hyperbolic sine is *number*, so ASINH(SINH(number)) equals *number*.

Syntax

```
ASINH (number)
```

The ASINH function syntax has the following arguments:

- Number Required. Any real number.

ATAN function

Description

Returns the arctangent, or inverse tangent, of a number. The arctangent is the angle whose tangent is *number*. The returned angle is given in radians in the range $-\pi/2$ to $\pi/2$.

Syntax

```
ATAN (number)
```

The ATAN function syntax has the following arguments:

- **Number** Required. The tangent of the angle you want.

Remark

To express the arctangent in degrees, multiply the result by $180/\pi$ () or use the DEGREES function.

ATAN2 function

Description

Returns the arctangent, or inverse tangent, of the specified x- and y-coordinates. The arctangent is the angle from the x-axis to a line containing the origin (0, 0) and a point with coordinates (x_num, y_num). The angle is given in radians between -pi and pi, excluding -pi.

Syntax

```
ATAN2(x_num, y_num)
```

The ATAN2 function syntax has the following arguments:

- X_num Required. The x-coordinate of the point.
- Y_num Required. The y-coordinate of the point.

Remarks

- A positive result represents a counterclockwise angle from the x-axis; a negative result represents a clockwise angle.
- $\text{ATAN2}(a,b)$ equals $\text{ATAN}(b/a)$, except that a can equal 0 in ATAN2.
- If both x_num and y_num are 0, ATAN2 returns the #DIV/0! error value.
- To express the arctangent in degrees, multiply the result by $180/\text{PI}()$ or use the DEGREES function.

ATANH function

Description

Returns the inverse hyperbolic tangent of a number. Number must be between -1 and 1 (excluding -1 and 1). The inverse hyperbolic tangent is the value whose hyperbolic tangent is *number*, so $\text{ATANH}(\text{TANH}(\text{number}))$ equals *number*.

Syntax

```
ATANH (number)
```

The ATANH function syntax has the following arguments:

- **Number** Required. Any real number between 1 and -1.

CEILING function

Description

Returns number rounded up, away from zero, to the nearest multiple of significance. For example, if you want to avoid using pennies in your prices and your product is priced at \$4.42, use the formula =CEILING(4.42,0.05) to round prices up to the nearest nickel.

Syntax

```
CEILING(number, significance)
```

The CEILING function syntax has the following arguments:

- **Number** Required. The value you want to round.
- **Significance** Required. The multiple to which you want to round.

Remarks

- If either argument is nonnumeric, CEILING returns the #VALUE! error value.
- Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an exact multiple of significance, no rounding occurs.
- If number is negative, and significance is negative, the value is rounded down, away from zero.
- If number is negative, and significance is positive, the value is rounded up towards zero.

CEILING.PRECISE function

Description

Returns a number that is rounded up to the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded up. However, if the number or the significance is zero, zero is returned.

Syntax

```
CEILING.PRECISE(number, [significance])
```

The CEILING.PRECISE function syntax has the following arguments:

- **Number** Required. The value to be rounded.
- **Significance** Optional. The multiple to which number is to be rounded.
- If significance is omitted, its default value is 1.

Remarks

- The absolute value of the multiple is used, so that the CEILING.PRECISE function returns the mathematical ceiling irrespective of the signs of number and significance.

COMBIN function

Description

Returns the number of combinations for a given number of items. Use COMBIN to determine the total possible number of groups for a given number of items.

Syntax

```
COMBIN(number, number_chosen)
```

The COMBIN function syntax has the following arguments:

- **Number** Required. The number of items.
- **Number_chosen** Required. The number of items in each combination.

Remarks

- Numeric arguments are truncated to integers.
- If either argument is nonnumeric, COMBIN returns the #VALUE! error value.
- If number < 0, number_chosen < 0, or number < number_chosen, COMBIN returns the #NUM! error value.
- A combination is any set or subset of items, regardless of their internal order. Combinations are distinct from permutations, for which the internal order is significant.
- The number of combinations is as follows, where number = n and number_chosen = k:

$$\binom{n}{k} = \frac{P_{k,n}}{k!} = \frac{n!}{k!(n-k)!}$$

where:

$$P_{k,n} = \frac{n!}{(n-k)!}$$

COS function

Description

Returns the cosine of the given angle.

Syntax

```
COS (number)
```

The COS function syntax has the following arguments:

- **Number** Required. The angle in radians for which you want the cosine.

Remark

If the angle is in degrees, either multiply the angle by $\text{PI}()/180$ or use the RADIANS function to convert the angle to radians.

COSH function

Description

Returns the hyperbolic cosine of a number.

Syntax

```
COSH (number)
```

The COSH function syntax has the following arguments:

- Number Required. Any real number for which you want to find the hyperbolic cosine.

Remark

The formula for the hyperbolic cosine is:

$$\text{COSH}(z) = \frac{e^z + e^{-z}}{2}$$

DEGREES function

Description

Converts radians into degrees.

Syntax

```
DEGREES (angle)
```

The DEGREES function syntax has the following arguments:

- **Angle** Required. The angle in radians that you want to convert.

EVEN function

Description

Returns number rounded up to the nearest even integer. You can use this function for processing items that come in twos. For example, a packing crate accepts rows of one or two items. The crate is full when the number of items, rounded up to the nearest two, matches the crate's capacity.

Syntax

```
EVEN (number)
```

The EVEN function syntax has the following arguments:

- **Number** Required. The value to round.

Remarks

- If number is nonnumeric, EVEN returns the #VALUE! error value.
- Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an even integer, no rounding occurs.

EXP function

Description

Returns e raised to the power of number. The constant e equals 2.71828182845904, the base of the natural logarithm.

Syntax

```
EXP (number)
```

The EXP function syntax has the following arguments:

- Number Required. The exponent applied to the base e.

Remarks

- To calculate powers of other bases, use the exponentiation operator (^).
- EXP is the inverse of LN, the natural logarithm of number.

FACT function

Description

Returns the factorial of a number. The factorial of a number is equal to $1*2*3*...*$ number.

Syntax

```
FACT (number)
```

The FACT function syntax has the following arguments:

- **Number** Required. The nonnegative number for which you want the factorial. If number is not an integer, it is truncated.

FACTDOUBLE function

Description

Returns the double factorial of a number.

Syntax

```
FACTDOUBLE (number)
```

The FACTDOUBLE function syntax has the following arguments:

- **Number** Required. The value for which to return the double factorial. If number is not an integer, it is truncated.

Remarks

- If number is nonnumeric, FACTDOUBLE returns the #VALUE! error value.
- If number is negative, FACTDOUBLE returns the #NUM! error value.

- If number is even:

$$n!! = n(n-2)(n-4)\dots(4)(2)$$

- If number is odd:

$$n!! = n(n-2)(n-4)\dots(3)(1)$$

FLOOR function

Description

Rounds number down, toward zero, to the nearest multiple of significance.

Syntax

```
FLOOR(number, significance)
```

The FLOOR function syntax has the following arguments:

- **Number** Required. The numeric value you want to round.
- **Significance** Required. The multiple to which you want to round.

Remarks

- If either argument is nonnumeric, FLOOR returns the #VALUE! error value.
- If number is positive and significance is negative, FLOOR returns the #NUM! error value.
- If the sign of number is positive, a value is rounded down and adjusted toward zero. If the sign of number is negative, a value is rounded down and adjusted away from zero. If number is an exact multiple of significance, no rounding occurs.

FLOOR.PRECISE function

Description

Returns a number that is rounded down to the nearest integer or to the nearest multiple of significance. Regardless of the sign of the number, the number is rounded down. However, if the number or the significance is zero, zero is returned.

Syntax

```
FLOOR.PRECISE(number, [significance])
```

The FLOOR.PRECISE function syntax has the following arguments:

- **Number** Required. The value to be rounded.
- **Significance** Optional. The multiple to which number is to be rounded.

If significance is omitted, its default value is 1.

Remarks

- The absolute value of the multiple is used, so that the FLOOR.PRECISE function returns the mathematical floor irrespective of the signs of number and significance.

GCD function

Description

Returns the greatest common divisor of two or more integers. The greatest common divisor is the largest integer that divides both number1 and number2 without a remainder.

Syntax

```
GCD(number1, [number2], ...)
```

The GCD function syntax has the following arguments:

- Number1, number2, ... Number1 is required, subsequent numbers are optional. 1 to 255 values. If any value is not an integer, it is truncated.

Remarks

- If any argument is nonnumeric, GCD returns the #VALUE! error value.
- If any argument is less than zero, GCD returns the #NUM! error value.
- One divides any value evenly.
- A prime number has only itself and one as even divisors.
- If a parameter to GCD is $\geq 2^{53}$, GCD returns the #NUM! error value.

INT function

Description

Rounds a number down to the nearest integer.

Syntax

```
INT (number)
```

The INT function syntax has the following arguments:

- **Number** Required. The real number you want to round down to an integer.

LCM function

Description

Returns the least common multiple of integers. The least common multiple is the smallest positive integer that is a multiple of all integer arguments number1, number2, and so on. Use LCM to add fractions with different denominators.

Syntax

```
LCM(number1, [number2], ...)
```

The LCM function syntax has the following arguments:

- Number1, number2,... Number1 is required, subsequent numbers are optional. 1 to 255 values for which you want the least common multiple. If value is not an integer, it is truncated.

Remarks

- If any argument is nonnumeric, LCM returns the #VALUE! error value.
- If any argument is less than zero, LCM returns the #NUM! error value.
- If $\text{LCM}(a,b) \geq 2^{53}$, LCM returns the #NUM! error value.

LN function

Description

Returns the natural logarithm of a number. Natural logarithms are based on the constant e (2.71828182845904).

Syntax

```
LN (number)
```

The LN function syntax has the following arguments:

- **Number** Required. The positive real number for which you want the natural logarithm.

Remark

LN is the inverse of the EXP function.

LOG function

Description

Returns the logarithm of a number to the base you specify.

Syntax

```
LOG(number, [base])
```

The LOG function syntax has the following arguments:

- **Number** Required. The positive real number for which you want the logarithm.
- **Base** Optional. The base of the logarithm. If base is omitted, it is assumed to be 10.

LOG10 function

Description

Returns the base-10 logarithm of a number.

Syntax

```
LOG10 (number)
```

The LOG10 function syntax has the following arguments:

- **Number** Required. The positive real number for which you want the base-10 logarithm.

MDETERM function

Description

Returns the matrix determinant of an array.

Syntax

```
MDETERM(array)
```

The MDETERM function syntax has the following arguments:

- **Array** Required. A numeric array with an equal number of rows and columns.

Remarks

- Array can be given as a cell range, for example, A1:C3; as an array constant, such as {1,2,3;4,5,6;7,8,9}; or as a name to either of these.
- MDETERM returns the #VALUE! error when:
- Any cells in array are empty or contain text.
- Array does not have an equal number of rows and columns.
- The matrix determinant is a number derived from the values in array. For a three-row, three-column array, A1:C3, the determinant is defined as:

MDETERM(A1:C3)

equals

$A1*(B2*C3-B3*C2) + A2*(B3*C1-B1*C3) + A3*(B1*C2-B2*C1)$

- Matrix determinants are generally used for solving systems of mathematical equations that involve several variables.
- MDETERM is calculated with an accuracy of approximately 16 digits, which may lead to a small numeric error when the calculation is not complete. For example, the determinant of a singular matrix may differ from zero by 1E-16.

MINVERSE function

Description

Returns the inverse matrix for the matrix stored in an array.

Syntax

```
MINVERSE(array)
```

The MINVERSE function syntax has the following arguments:

- **Array** Required. A numeric array with an equal number of rows and columns.

Remarks

- Array can be given as a cell range, such as A1:C3; as an array constant, such as {1,2,3;4,5,6;7,8,9}; or as a name for either of these.
- If any cells in array are empty or contain text, MINVERSE returns the #VALUE! error value.
- MINVERSE also returns the #VALUE! error value if array does not have an equal number of rows and columns.
- Formulas that return arrays must be entered as array formulas.
- Inverse matrices, like determinants, are generally used for solving systems of mathematical equations involving several variables. The product of a matrix and its inverse is the identity matrix — the square array in which the diagonal values equal 1, and all other values equal 0.
- As an example of how a two-row, two-column matrix is calculated, suppose that the range A1:B2 contains the letters a, b, c, and d that represent any four numbers. The following table shows the inverse of the matrix A1:B2.

	Column A	Column B
Row 1	$d/(a*d-b*c)$	$b/(b*c-a*d)$
Row 2	$c/(b*c-a*d)$	$a/(a*d-b*c)$

- MINVERSE is calculated with an accuracy of approximately 16 digits, which may lead to a small numeric error when the cancellation is not complete.
- Some square matrices cannot be inverted and will return the #NUM! error value with MINVERSE. The determinant for a noninvertable matrix is 0.

MMULT function

Description

Returns the matrix product of two arrays. The result is an array with the same number of rows as array1 and the same number of columns as array2.

Syntax

```
MMULT(array1, array2)
```

The MMULT function syntax has the following arguments:

- Array1, array2 Required. The arrays you want to multiply.

Remarks

- The number of columns in array1 must be the same as the number of rows in array2, and both arrays must contain only numbers.
- Array1 and array2 can be given as cell ranges, array constants, or references.
- MMULT returns the #VALUE! error when:
 - Any cells are empty or contain text.
 - The number of columns in array1 is different from the number of rows in array2.
- The matrix product array a of two arrays b and c is:

$$a_{ji} = \sum_{k=1}^n b_{jk} c_{ki}$$

where i is the row number, and j is the column number.

- Formulas that return arrays must be entered as array formulas.

MOD function

Description

Returns the remainder after number is divided by divisor. The result has the same sign as divisor.

Syntax

```
MOD(number, divisor)
```

The MOD function syntax has the following arguments:

- **Number** Required. The number for which you want to find the remainder.
- **Divisor** Required. The number by which you want to divide number.

Remarks

- If divisor is 0, MOD returns the #DIV/0! error value.
- The MOD function can be expressed in terms of the INT function:

```
MOD(n, d) = n - d*INT(n/d)
```

MROUND function

Description

Returns a number rounded to the desired multiple.

Syntax

```
MROUND(number, multiple)
```

The MROUND function syntax has the following arguments:

- **Number** Required. The value to round.
- **Multiple** Required. The multiple to which you want to round number.

Remark

MROUND rounds up, away from zero, if the remainder of dividing number by multiple is greater than or equal to half the value of multiple.

MULTINOMIAL function

Description

Returns the ratio of the factorial of a sum of values to the product of factorials.

Syntax

```
MULTINOMIAL(number1, [number2], ...)
```

The MULTINOMIAL function syntax has the following arguments:

- Number1, number2, ... Number1 is required, subsequent numbers are optional. 1 to 255 values for which you want the multinomial.

Remarks

- If any argument is nonnumeric, MULTINOMIAL returns the #VALUE! error value.
- If any argument is less than zero, MULTINOMIAL returns the #NUM! error value.
- The multinomial is:

$$\text{MULTINOMIAL}(a,b,c) = \frac{(a+b+c)!}{a!b!c!}$$

ODD function

Description

Returns number rounded up to the nearest odd integer.

Syntax

```
ODD (number)
```

The ODD function syntax has the following arguments:

- **Number** Required. The value to round.

Remarks

- If number is nonnumeric, ODD returns the #VALUE! error value.
- Regardless of the sign of number, a value is rounded up when adjusted away from zero. If number is an odd integer, no rounding occurs.

PI function

Description

Returns the number 3.14159265358979, the mathematical constant pi, accurate to 15 digits.

Syntax

```
PI ()
```

The PI function syntax has no arguments:

POWER function

Description

Returns the result of a number raised to a power.

Syntax

```
POWER(number, power)
```

The POWER function syntax has the following arguments:

- **Number** Required. The base number. It can be any real number.
- **Power** Required. The exponent to which the base number is raised.

Remark

The "^" operator can be used instead of POWER to indicate to what power the base number is to be raised, such as in 5^2.

PRODUCT function

Description

The **PRODUCT** function multiplies all the numbers given as arguments and returns the product. For example, if cells A1 and A2 contain numbers, you can use the formula **=PRODUCT(A1, A2)** to multiply those two numbers together. You can also perform the same operation by using the multiply (*) mathematical operator; for example, **=A1 * A2**.

The **PRODUCT** function is useful when you need to multiply many cells together. For example, the formula **=PRODUCT(A1:A3, C1:C3)** is equivalent to **=A1 * A2 * A3 * C1 * C2 * C3**.

Syntax

```
PRODUCT(number1, [number2], ...)
```

The **PRODUCT** function syntax has the following arguments:

- **number1** Required. The first number or range (range: Two or more cells on a sheet. The cells in a range can be adjacent or nonadjacent.) that you want to multiply.
- **number2, ...** Optional. Additional numbers or ranges that you want to multiply, up to a maximum of 255 arguments.

NOTE If an argument is an array or reference, only numbers in the array or reference are multiplied. Empty cells, logical values, and text in the array or reference are ignored.

QUOTIENT function

Description

Returns the integer portion of a division. Use this function when you want to discard the remainder of a division.

Syntax

```
QUOTIENT(numerator, denominator)
```

The QUOTIENT function syntax has the following arguments:

- Numerator Required. The dividend.
- Denominator Required. The divisor.

Remark

If either argument is nonnumeric, QUOTIENT returns the #VALUE! error value.

RADIANS function

Description

Converts degrees to radians.

Syntax

```
RADIANS (angle)
```

The RADIANS function syntax has the following arguments:

- **Angle** Required. An angle in degrees that you want to convert.

RAND function

Description

Returns an evenly distributed random real number greater than or equal to 0 and less than 1. A new random real number is returned every time the worksheet is calculated.

Syntax

```
RAND ( )
```

The RAND function syntax has no arguments.

Remarks

- To generate a random real number between a and b, use:
- $\text{RAND()}*(b-a)+a$
- If you want to use RAND to generate a random number but don't want the numbers to change every time the cell is calculated, you can enter `=RAND()` in the formula bar, and then press F9 to change the formula to a random number.

RANDBETWEEN function

Description

Returns a random integer number between the numbers you specify. A new random integer number is returned every time the worksheet is calculated.

Syntax

```
RANDBETWEEN(bottom, top)
```

The RANDBETWEEN function syntax has the following arguments:

- **Bottom** Required. The smallest integer RANDBETWEEN will return.
- **Top** Required. The largest integer RANDBETWEEN will return.

ROMAN function

Description

Converts an arabic numeral to roman, as text.

Syntax

```
ROMAN(number, [form])
```

The ROMAN function syntax has the following arguments:

- **Number** Required. The Arabic numeral you want converted.
- **Form** Optional. A number specifying the type of roman numeral you want. The roman numeral style ranges from Classic to Simplified, becoming more concise as the value of form increases.

See the example following

```
ROMAN(499,0)
```

below.

Form	Type
0 or omitted	Classic.
1	More concise. See example below.
2	More concise. See example below.
3	More concise. See example below.
4	Simplified.
TRUE	Classic.
FALSE	Simplified.

Remarks

- If number is negative, the #VALUE! error value is returned.
- If number is greater than 3999, the #VALUE! error value is returned.

ROUND function

Description

The **ROUND** function rounds a number to a specified number of digits. For example, if cell A1 contains 23.7825, and you want to round that value to two decimal places, you can use the following formula:

```
=ROUND(A1, 2)
```

The result of this function is 23.78.

Syntax

```
ROUND(number, num_digits)
```

The ROUND function syntax has the following arguments:

- **Number** Required. The number that you want to round.
- **num_digits** Required. The number of digits to which you want to round the number argument.

Remarks

- If num_digits is greater than 0 (zero), then number is rounded to the specified number of decimal places.
- If num_digits is 0, the number is rounded to the nearest integer.
- If num_digits is less than 0, the number is rounded to the left of the decimal point.
- To always round up (away from zero), use the ROUNDUP function.
- To always round down (toward zero), use the ROUNDOWN function.
- To round a number to a specific multiple (for example, to round to the nearest 0.5), use the MROUND function.

ROUNDDOWN function

Description

Rounds a number down, toward zero.

Syntax

```
ROUNDDOWN(number, num_digits)
```

The ROUNDDOWN function syntax has the following arguments:

- **Number** Required. Any real number that you want rounded down.
- **Num_digits** Required. The number of digits to which you want to round number.

Remarks

- ROUNDDOWN behaves like ROUND, except that it always rounds a number down.
- If num_digits is greater than 0 (zero), then number is rounded down to the specified number of decimal places.
- If num_digits is 0, then number is rounded down to the nearest integer.
- If num_digits is less than 0, then number is rounded down to the left of the decimal point.

ROUNDUP function

Description

Rounds a number up, away from 0 (zero).

Syntax

```
ROUNDUP(number, num_digits)
```

The ROUNDUP function syntax has the following arguments:

- **Number** Required. Any real number that you want rounded up.
- **Num_digits** Required. The number of digits to which you want to round number.

Remarks

- ROUNDUP behaves like ROUND, except that it always rounds a number up.
- If num_digits is greater than 0 (zero), then number is rounded up to the specified number of decimal places.
- If num_digits is 0, then number is rounded up to the nearest integer.
- If num_digits is less than 0, then number is rounded up to the left of the decimal point.

SERIESSUM function

Description

Many functions can be approximated by a power series expansion.

Returns the sum of a power series based on the formula:

$$\text{SERIES}(x, n, m, a) = a_1x^n + a_2x^{(n+m)} + a_3x^{(n+2m)} \\ + \dots + a_jx^{(n+(j-1)m)}$$

Syntax

```
SERIESSUM(x, n, m, coefficients)
```

The SERIESSUM function syntax has the following arguments:

- **X** Required. The input value to the power series.
- **N** Required. The initial power to which you want to raise x.
- **M** Required. The step by which to increase n for each term in the series.
- **Coefficients** Required. A set of coefficients by which each successive power of x is multiplied. The number of values in coefficients determines the number of terms in the power series. For example, if there are three values in coefficients, then there will be three terms in the power series.

Remark

If any argument is nonnumeric, SERIESSUM returns the #VALUE! error value.

SIGN function

Description

Determines the sign of a number. Returns 1 if the number is positive, zero (0) if the number is 0, and -1 if the number is negative.

Syntax

```
SIGN (number)
```

The SIGN function syntax has the following arguments:

- Number Required. Any real number.

SIN function

Description

Returns the sine of the given angle.

Syntax

```
SIN (number)
```

The SIN function syntax has the following arguments:

- **Number** Required. The angle in radians for which you want the sine.

Remark

If your argument is in degrees, multiply it by $\text{PI}()/180$ or use the RADIANS function to convert it to radians.

SINH function

Description

Returns the hyperbolic sine of a number.

Syntax

```
SINH (number)
```

The SINH function syntax has the following arguments:

- Number Required. Any real number.

Remark

The formula for the hyperbolic sine is:

$$\text{SINH}(z) = \frac{e^z - e^{-z}}{2}$$

SQRT function

Description

Returns a positive square root.

Syntax

```
SQRT (number)
```

The SQRT function syntax has the following arguments:

- **Number** Required. The number for which you want the square root.

Remark

If number is negative, SQRT returns the #NUM! error value.

SQRTPI function

Description

Returns the square root of (number * pi).

Syntax

```
SQRTPI (number)
```

The SQRTPI function syntax has the following arguments:

- **Number** Required. The number by which pi is multiplied.

Remark

If number < 0, SQRTPI returns the #NUM! error value.

SUBTOTAL function

Description

Returns a subtotal in a list or database. It is generally easier to create a list with subtotals by using the **Subtotal** command in the **Outline** group on the **Data** tab. Once the subtotal list is created, you can modify it by editing the SUBTOTAL function.

Syntax

```
SUBTOTAL(function_num,ref1,[ref2],...)
```

The SUBTOTAL function syntax has the following arguments:

- **Function_num** Required. The number 1 to 11 (includes hidden values) or 101 to 111 (ignores hidden values) that specifies which function to use in calculating subtotals within a list.

Function_num (includes hidden values)	Function_num (ignores hidden values)	Function
1	101	AVERAGE
2	102	COUNT
3	103	COUNTA
4	104	MAX
5	105	MIN
6	106	PRODUCT
7	107	STDEV
8	108	STDEVP
9	109	SUM
10	110	VAR
11	111	VARP

- Ref1 Required. The first named range or reference for which you want the subtotal.
- Ref2,... Optional. Named ranges or references 2 to 254 for which you want the subtotal.

Remarks

- If there are other subtotals within ref1, ref2,... (or nested subtotals), these nested subtotals are ignored to avoid double counting.
- For the function_num constants from 1 to 11, the SUBTOTAL function includes the values of rows hidden by the Hide Rows command under the Hide & Unhide submenu of the Format command in the Cells group on the Home tab. Use these constants when you want to subtotal hidden and nonhidden numbers in a list. For the function_Num constants from 101 to 111, the SUBTOTAL function ignores values of rows hidden by the Hide Rows command. Use these constants when you want to subtotal only nonhidden numbers in a list.
- The SUBTOTAL function ignores any rows that are not included in the result of a filter, no matter which function_num value you use.
- The SUBTOTAL function is designed for columns of data, or vertical ranges. It is not designed for rows of data, or horizontal ranges. For example, when you subtotal a horizontal range using a function_num of 101 or greater, such as SUBTOTAL(109,B2:G2), hiding a column does not affect the subtotal. But, hiding a row in a subtotal of a vertical range does affect the subtotal.
- If any of the references are 3-D references, SUBTOTAL returns the #VALUE! error value.

SUM function

Description

The **SUM** function adds all the numbers that you specify as **arguments**. Each argument can be a **range**, a **cell reference**, an **array**, a **constant**, a **formula**, or the result from another function. For example, **SUM(A1:A5)** adds all the numbers that are contained in cells A1 through A5. For another example, **SUM(A1, A3, A5)** adds the numbers that are contained in cells A1, A3, and A5.

Syntax

```
SUM(number1, [number2], ...)
```

The SUM function syntax has the following arguments:

- **number1** Required. The first number argument that you want to add.
- **number2,...** Optional. Number arguments 2 to 255 that you want to add.

Remarks

- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, or text in the array or reference are ignored.
- If any arguments are error values, or if any arguments are text that cannot be translated into numbers, displays an error.

SUMIF function

Description

You use the **SUMIF** function to sum the values in a **range** that meet criteria that you specify. For example, suppose that in a column that contains numbers, you want to sum only the values that are larger than 5. You can use the following formula:

```
=SUMIF(B2:B25, ">5")
```

In this example, the criteria is applied the same values that are being summed. If you want, you can apply the criteria to one range and sum the corresponding values in a different range. For example, the formula **=SUMIF(B2:B5, "John", C2:C5)** sums only the values in the range C2:C5, where the corresponding cells in the range B2:B5 equal "John."

NOTE To sum cells based on multiple criteria, see [SUMIFS function](#).

Syntax

```
SUMIF(range, criteria, [sum_range])
```

The **SUMIF** function syntax has the following arguments:

- **range** Required. The range of cells that you want evaluated by criteria. Cells in each range must be numbers or names, arrays, or references that contain numbers. Blank and text values are ignored.
- **criteria** Required. The criteria in the form of a number, expression, a cell reference, text, or a function that defines which cells will be added. For example, criteria can be expressed as 32, ">32", B5, 32, "32", "apples", or TODAY().
- **Important** Any text criteria or any criteria that includes logical or mathematical symbols must be enclosed in double quotation marks ("). If the criteria is numeric, double quotation marks are not required.

- `sum_range` Optional. The actual cells to add, if you want to add cells other than those specified in the range argument. If the `sum_range` argument is omitted, Adds the cells that are specified in the range argument (the same cells to which the criteria is applied).

NOTES

- The `sum_range` argument does not have to be the same size and shape as the range argument. The actual cells that are added are determined by using the upper leftmost cell in the `sum_range` argument as the beginning cell, and then including cells that correspond in size and shape to the range argument. For example:

If range is	And <code>sum_range</code> is	Then the actual cells are
A1:A5	B1:B5	B1:B5
A1:A5	B1:B3	B1:B5
A1:B4	C1:D4	C1:D4
A1:B4	C1:C2	C1:D4

- You can use the wildcard characters — the question mark (?) and asterisk (*) — as the criteria argument. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) preceding the character.

SUMIFS function

Description

Adds the cells in a **range** that meet multiple criteria. For example, if you want to sum the numbers in the range A1:A20 only if the corresponding numbers in B1:B20 are greater than zero (0) and the corresponding numbers in C1:C20 are less than 10, you can use the following formula:

```
=SUMIFS(A1:A20, B1:B20, ">0", C1:C20, "<10")
```

IMPORTANT The order of arguments differ between the **SUMIFS** and **SUMIF** functions. In particular, the **sum_range** argument is the first argument in **SUMIFS**, but it is the third argument in **SUMIF**. If you are copying and editing these similar functions, make sure you put the arguments in the correct order.

Syntax

```
SUMIFS(sum_range, criteria_range1, criteria1, [criteria_range2, criteria2], ...)
```

The SUMIFS function syntax has the following arguments:

- **sum_range** Required. One or more cells to sum, including numbers or names, ranges, or cell references (cell reference: The set of coordinates that a cell occupies on a worksheet. For example, the reference of the cell that appears at the intersection of column B and row 3 is B3.) that contain numbers. Blank and text values are ignored.
- **criteria_range1** Required. The first range in which to evaluate the associated criteria.
- **criteria1** Required. The criteria in the form of a number, expression, cell reference, or text that define which cells in the **criteria_range1** argument will be added. For example, criteria can be expressed as 32, ">32", B4, "apples", or "32."
- **criteria_range2, criteria2, ...** Optional. Additional ranges and their associated criteria. Up to 127 range/criteria pairs are allowed.

Remarks

- Each cell in the sum_range argument is summed only if all of the corresponding criteria specified are true for that cell. For example, suppose that a formula contains two criteria_range arguments. If the first cell of criteria_range1 meets criteria1, and the first cell of criteria_range2 meets criteria2, the first cell of sum_range is added to the sum, and so on, for the remaining cells in the specified ranges.
- Cells in the sum_range argument that contain TRUE evaluate to 1; cells in sum_range that contain FALSE evaluate to 0 (zero).
- Unlike the range and criteria arguments in the SUMIF function, in the SUMIFS function, each criteria_range argument must contain the same number of rows and columns as the sum_range argument.
- You can use the wildcard characters — the question mark (?) and asterisk (*) — in criteria. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.

SUMPRODUCT function

Description

Multiplies corresponding components in the given arrays, and returns the sum of those products.

Syntax

```
SUMPRODUCT(array1, [array2], [array3], ...)
```

The SUMPRODUCT function syntax has the following arguments:

- **Array1** Required. The first array argument whose components you want to multiply and then add.
- **Array2, array3,...** Optional. Array arguments 2 to 255 whose components you want to multiply and then add.

Remarks

- The array arguments must have the same dimensions. If they do not, SUMPRODUCT returns the #VALUE! error value.
- SUMPRODUCT treats array entries that are not numeric as if they were zeros.

SUMSQ function

Description

Returns the sum of the squares of the arguments.

Syntax

```
SUMSQ(number1, [number2], ...)
```

The SUMSQ function syntax has the following arguments:

- Number1, number2, ... Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want the sum of the squares. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Numbers, logical values, and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.

SUMX2MY2 function

Description

Returns the sum of the difference of squares of corresponding values in two arrays.

Syntax

```
SUMX2MY2(array_x, array_y)
```

The SUMX2MY2 function syntax has the following arguments:

- **Array_x** Required. The first array or range of values.
- **Array_y** Required. The second array or range of values.

Remarks

- The arguments should be either numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array_x and array_y have a different number of values, SUMX2MY2 returns the #N/A error value.
- The equation for the sum of the difference of squares is:

$$\text{SUMX2MY2} = \sum (x^2 - y^2)$$

SUMX2PY2 function

Description

Returns the sum of the sum of squares of corresponding values in two arrays. The sum of the sum of squares is a common term in many statistical calculations.

Syntax

```
SUMX2PY2(array_x, array_y)
```

The SUMX2PY2 function syntax has the following arguments:

- **Array_x** Required. The first array or range of values.
- **Array_y** Required. The second array or range of values.

Remarks

- The arguments should be either numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array_x and array_y have a different number of values, SUMX2PY2 returns the #N/A error value.
- The equation for the sum of the sum of squares is:

$$\text{SUMX2PY2} = \sum (x^2 + y^2)$$

SUMXMY2 function

Description

Returns the sum of squares of differences of corresponding values in two arrays.

Syntax

```
SUMXMY2(array_x, array_y)
```

The SUMXMY2 function syntax has the following arguments:

- **Array_x** Required. The first array or range of values.
- **Array_y** Required. The second array or range of values.

Remarks

- The arguments should be either numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array_x and array_y have a different number of values, SUMXMY2 returns the #N/A error value.
- The equation for the sum of squared differences is:

$$\text{SUMXMY2} = \sum (x - y)^2$$

TAN function

Description

Returns the tangent of the given angle.

Syntax

```
TAN (number)
```

The TAN function syntax has the following arguments:

- **Number** Required. The angle in radians for which you want the tangent.

Remark

If your argument is in degrees, multiply it by $\text{PI}()/180$ or use the RADIANS function to convert it to radians.

TANH function

Description

Returns the hyperbolic tangent of a number.

Syntax

```
TANH (number)
```

The TANH function syntax has the following arguments:

- Number Required. Any real number.

Remark

The formula for the hyperbolic tangent is:

$$\text{TANH}(z) = \frac{\text{SINH}(z)}{\text{COSH}(z)}$$

TRUNC function

Description

Truncates a number to an integer by removing the fractional part of the number.

Syntax

```
TRUNC (number, [num_digits])
```

The TRUNC function syntax has the following arguments:

- **Number** Required. The number you want to truncate.
- **Num_digits** Optional. A number specifying the precision of the truncation. The default value for num_digits is 0 (zero).

Remark

TRUNC and INT are similar in that both return integers. TRUNC removes the fractional part of the number. INT rounds numbers down to the nearest integer based on the value of the fractional part of the number. INT and TRUNC are different only when using negative numbers:

```
TRUNC (-4.3)
```

returns -4, but

```
INT (-4.3)
```

returns -5 because -5 is the lower number.

AVEDEV function

Description

Returns the average of the absolute deviations of data points from their mean. AVEDEV is a measure of the variability in a data set.

Syntax

```
AVEDEV(number1, [number2], ...)
```

The AVEDEV function syntax has the following arguments:

- Number1, number2, ... Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want the average of the absolute deviations. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- AVEDEV is influenced by the unit of measurement in the input data.
- Arguments must either be numbers or be names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- The equation for average deviation is:

$$\frac{1}{n} \sum |x - \bar{x}|$$

AVERAGE function

Description

Returns the average (arithmetic mean) of the arguments. For example, if the **range** A1:A20 contains numbers, the formula **=AVERAGE(A1:A20)** returns the average of those numbers.

Syntax

```
AVERAGE(number1, [number2], ...)
```

The AVERAGE function syntax has the following arguments:

- **Number1** Required. The first number, cell reference (cell reference: The set of coordinates that a cell occupies on a worksheet. For example, the reference of the cell that appears at the intersection of column B and row 3 is B3.), or range for which you want the average.
- **Number2, ...** Optional. Additional numbers, cell references or ranges for which you want the average, up to a maximum of 255.

Remarks

- Arguments can either be numbers or names, ranges, or cell references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If a range or cell reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the AVERAGEA function.
- If you want to calculate the average of only the values that meet certain criteria, use the AVERAGEIF function or the AVERAGEIFS function.

NOTE The **AVERAGE** function measures central tendency, which is the location of the center of a group of numbers in a statistical distribution. The three most common measures of central tendency are:

- Average, which is the arithmetic mean, and is calculated by adding a group of numbers and then dividing by the count of those numbers. For example, the average of 2, 3, 3, 5, 7, and 10 is 30 divided by 6, which is 5.
- Median, which is the middle number of a group of numbers; that is, half the numbers have values that are greater than the median, and half the numbers have values that are less than the median. For example, the median of 2, 3, 3, 5, 7, and 10 is 4.
- Mode, which is the most frequently occurring number in a group of numbers. For example, the mode of 2, 3, 3, 5, 7, and 10 is 3.

For a symmetrical distribution of a group of numbers, these three measures of central tendency are all the same. For a skewed distribution of a group of numbers, they can be different.

AVERAGEA function

Description

Calculates the average (arithmetic mean) of the values in the list of arguments.

Syntax

```
AVERAGEA(value1, [value2], ...)
```

The AVERAGEA function syntax has the following arguments:

- Value1, value2, ... Value1 is required, subsequent values are optional. 1 to 255 cells, ranges of cells, or values for which you want the average.

Remarks

- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- Arguments that contain TRUE evaluate as 1; arguments that contain FALSE evaluate as 0 (zero).
- Array or reference arguments that contain text evaluate as 0 (zero). Empty text ("") evaluates as 0 (zero).
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the AVERAGE function.

NOTE The AVERAGEA function measures central tendency, which is the location of the center of a group of numbers in a statistical distribution. The three most common measures of central tendency are:

- Average which is the arithmetic mean, and is calculated by adding a group of numbers and then dividing by the count of those numbers. For example, the average of 2, 3, 3, 5, 7, and 10 is 30 divided by 6, which is 5.
- Median which is the middle number of a group of numbers; that is, half the numbers have values that are greater than the median, and half the numbers have values that are less than the median. For example, the median of 2, 3, 3, 5, 7, and 10 is 4.
- Mode which is the most frequently occurring number in a group of numbers. For example, the mode of 2, 3, 3, 5, 7, and 10 is 3.

For a symmetrical distribution of a group of numbers, these three measures of central tendency are all the same. For a skewed distribution of a group of numbers, they can be different.

AVERAGEIF function

Description

Returns the average (arithmetic mean) of all the cells in a range that meet a given criteria.

Syntax

```
AVERAGEIF(range, criteria, [average_range])
```

The AVERAGEIF function syntax has the following arguments:

- **Range** Required. One or more cells to average, including numbers or names, arrays, or references that contain numbers.
- **Criteria** Required. The criteria in the form of a number, expression, cell reference, or text that defines which cells are averaged. For example, criteria can be expressed as 32, "32", ">32", "apples", or B4.
- **Average_range** Optional. The actual set of cells to average. If omitted, range is used.

Remarks

- Cells in range that contain TRUE or FALSE are ignored.
- If a cell in average_range is an empty cell, AVERAGEIF ignores it.
- If range is a blank or text value, AVERAGEIF returns the #DIV/0! error value.
- If a cell in criteria is empty, AVERAGEIF treats it as a 0 value.
- If no cells in the range meet the criteria, AVERAGEIF returns the #DIV/0! error value.
- You can use the wildcard characters, question mark (?) and asterisk (*), in criteria. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.
- Average_range does not have to be the same size and shape as range. The actual cells that are averaged are determined by using the top, left cell in average_range as the beginning cell, and then including cells that correspond in size and shape to range. For example:

If range is	And average_range is	Then the actual cells evaluated are
A1:A5	B1:B5	B1:B5
A1:A5	B1:B3	B1:B5
A1:B4	C1:D4	C1:D4
A1:B4	C1:C2	C1:D4

NOTE The AVERAGEIF function measures central tendency, which is the location of the center of a group of numbers in a statistical distribution. The three most common measures of central tendency are:

- **Average** which is the arithmetic mean, and is calculated by adding a group of numbers and then dividing by the count of those numbers. For example, the average of 2, 3, 3, 5, 7, and 10 is 30 divided by 6, which is 5.
- **Median** which is the middle number of a group of numbers; that is, half the numbers have values that are greater than the median, and half the numbers have values that are less than the median. For example, the median of 2, 3, 3, 5, 7, and 10 is 4.
- **Mode** which is the most frequently occurring number in a group of numbers. For example, the mode of 2, 3, 3, 5, 7, and 10 is 3.

For a symmetrical distribution of a group of numbers, these three measures of central tendency are all the same. For a skewed distribution of a group of numbers, they can be different.

AVERAGEIFS function

Description

Returns the average (arithmetic mean) of all cells that meet multiple criteria.

Syntax

```
AVERAGEIFS(average_range, criteria_range1, criteria1, [criteria_range2, criteria2], ...)
```

The AVERAGEIFS function syntax has the following arguments:

- **Average_range** Required. One or more cells to average, including numbers or names, arrays, or references that contain numbers.
- **Criteria_range1, criteria_range2, ...** Criteria_range1 is required, subsequent criteria_ranges are optional. 1 to 127 ranges in which to evaluate the associated criteria.
- **Criteria1, criteria2, ...** Criteria1 is required, subsequent criteria are optional. 1 to 127 criteria in the form of a number, expression, cell reference, or text that define which cells will be averaged. For example, criteria can be expressed as 32, "32", ">32", "apples", or B4.

Remarks

- If **average_range** is a blank or text value, AVERAGEIFS returns the #DIV0! error value.
- If a cell in a criteria range is empty, AVERAGEIFS treats it as a 0 value.
- Cells in range that contain TRUE evaluate as 1; cells in range that contain FALSE evaluate as 0 (zero).
- Each cell in **average_range** is used in the average calculation only if all of the corresponding criteria specified are true for that cell.
- Unlike the range and criteria arguments in the AVERAGEIF function, in AVERAGEIFS each **criteria_range** must be the same size and shape as **sum_range**.

- If cells in average_range cannot be translated into numbers, AVERAGEIFS returns the #DIV/0! error value.
- If there are no cells that meet all the criteria, AVERAGEIFS returns the #DIV/0! error value.
- You can use the wildcard characters, question mark (?) and asterisk (*), in criteria. A question mark matches any single character; an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.

NOTE The AVERAGEIFS function measures central tendency, which is the location of the center of a group of numbers in a statistical distribution. The three most common measures of central tendency are:

- **Average** which is the arithmetic mean, and is calculated by adding a group of numbers and then dividing by the count of those numbers. For example, the average of 2, 3, 3, 5, 7, and 10 is 30 divided by 6, which is 5.
- **Median** which is the middle number of a group of numbers; that is, half the numbers have values that are greater than the median, and half the numbers have values that are less than the median. For example, the median of 2, 3, 3, 5, 7, and 10 is 4.
- **Mode** which is the most frequently occurring number in a group of numbers. For example, the mode of 2, 3, 3, 5, 7, and 10 is 3.

For a symmetrical distribution of a group of numbers, these three measures of central tendency are all the same. For a skewed distribution of a group of numbers, they can be different.

BETA.DIST function

Description

Returns the beta distribution.

The beta distribution is commonly used to study variation in the percentage of something across samples, such as the fraction of the day people spend watching television.

Syntax

```
BETA.DIST(x, alpha, beta, cumulative, [A], [B])
```

The BETA.DIST function syntax has the following arguments:

- **X** Required. The value between A and B at which to evaluate the function
- **Alpha** Required. A parameter of the distribution.
- **Beta** Required. A parameter of the distribution.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, BETA.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.
- **A** Optional. A lower bound to the interval of x.
- **B** Optional. An upper bound to the interval of x.

Remarks

- If any argument is nonnumeric, BETA.DIST returns the #VALUE! error value.
- If $\alpha \leq 0$ or $\beta \leq 0$, BETA.DIST returns the #NUM! error value.
- If $x < A$, $x > B$, or $A = B$, BETA.DIST returns the #NUM! error value.
- If you omit values for A and B, BETA.DIST uses the standard cumulative beta distribution, so that $A = 0$ and $B = 1$

BETA.INV function

Description

Returns the inverse of the beta cumulative probability density function (BETA.DIST).

If $\text{probability} = \text{BETA.DIST}(x, \dots, \text{TRUE})$, then $\text{BETA.INV}(\text{probability}, \dots) = x$. The beta distribution can be used in project planning to model probable completion times given an expected completion time and variability.

Syntax

```
BETA.INV(probability, alpha, beta, [A], [B])
```

The BETA.INV function syntax has the following arguments:

- **Probability** Required. A probability associated with the beta distribution.
- **Alpha** Required. A parameter of the distribution.
- **Beta** Required. A parameter the distribution.
- **A** Optional. A lower bound to the interval of x.
- **B** Optional. An upper bound to the interval of x.

Remarks

- If any argument is nonnumeric, BETA.INV returns the #VALUE! error value.
- If $\alpha \leq 0$ or $\beta \leq 0$, BETA.INV returns the #NUM! error value.
- If $\text{probability} \leq 0$ or $\text{probability} > 1$, BETA.INV returns the #NUM! error value.
- If you omit values for A and B, BETA.INV uses the standard cumulative beta distribution, so that $A = 0$ and $B = 1$.

Given a value for probability, BETA.INV seeks that value x such that $\text{BETA.DIST}(x, \alpha, \beta, \text{TRUE}, A, B) = \text{probability}$. Thus, precision of BETA.INV depends on precision of BETA.DIST.

BINOM.DIST function

Description

Returns the individual term binomial distribution probability. Use BINOM.DIST in problems with a fixed number of tests or trials, when the outcomes of any trial are only success or failure, when trials are independent, and when the probability of success is constant throughout the experiment. For example, BINOM.DIST can calculate the probability that two of the next three babies born are male.

Syntax

```
BINOM.DIST(number_s, trials, probability_s, cumulative)
```

The BINOM.DIST function syntax has the following arguments:

- **Number_s** Required. The number of successes in trials.
- **Trials** Required. The number of independent trials.
- **Probability_s** Required. The probability of success on each trial.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, then BINOM.DIST returns the cumulative distribution function, which is the probability that there are at most number_s successes; if FALSE, it returns the probability mass function, which is the probability that there are number_s successes.

Remarks

- Number_s and trials are truncated to integers.
- If number_s, trials, or probability_s is nonnumeric, BINOM.DIST returns the #VALUE! error value.
- If number_s < 0 or number_s > trials, BINOM.DIST returns the #NUM! error value.
- If probability_s < 0 or probability_s > 1, BINOM.DIST returns the #NUM! error value.
- If x = number_s, n = trials, and p = probability_s, then the binomial probability mass function is:

$$b(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}$$

where:

$$\binom{n}{x}$$

is COMBIN(n,x).

- If $x = \text{number_s}$, $n = \text{trials}$, and $p = \text{probability_s}$, then the cumulative binomial distribution is:

$$B(x, n, p) = \sum_{y=0}^x b(y, n, p)$$

BINOM.INV function

Description

Returns the smallest value for which the cumulative binomial distribution is greater than or equal to a criterion value.

Syntax

```
BINOM.INV(trials,probability_s,alpha)
```

The BINOM.INV function syntax has the following arguments:

Trials Required. The number of Bernoulli trials.

Probability_s Required. The probability of a success on each trial.

Alpha Required. The criterion value.

Remarks

- If any argument is nonnumeric, BINOM.INV returns the #VALUE! error value.
- If trials is not an integer, it is truncated.
- If trials < 0, BINOM.INV returns the #NUM! error value.
- If probability_s is < 0 or probability_s > 1, BINOM.INV returns the #NUM! error value.
- If alpha < 0 or alpha > 1, BINOM.INV returns the #NUM! error value.

CHISQ.DIST function

Description

Returns the chi-squared distribution.

The chi-squared distribution is commonly used to study variation in the percentage of something across samples, such as the fraction of the day people spend watching television.

Syntax

```
CHISQ.DIST(x, deg_freedom, cumulative)
```

The CHISQ.DIST function syntax has the following arguments:

- **X** Required. The value at which you want to evaluate the distribution.
- **Deg_freedom** Required. The number of degrees of freedom.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, CHISQ.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- If any argument is nonnumeric, CHISQ.DIST returns the #VALUE! error value.
- If x is negative, CHISQ.DIST returns the #NUM! error value.
- If deg_freedom is not an integer, it is truncated.
- If deg_freedom < 1 or deg_freedom > 10¹⁰, CHISQ.DIST returns the #NUM! error value.

CHISQ.DIST.RT function

Description

Returns the right-tailed probability of the chi-squared distribution.

The χ^2 distribution is associated with a χ^2 test. Use the χ^2 test to compare observed and expected values. For example, a genetic experiment might hypothesize that the next generation of plants will exhibit a certain set of colors. By comparing the observed results with the expected ones, you can decide whether your original hypothesis is valid.

Syntax

```
CHISQ.DIST.RT(x, deg_freedom)
```

The CHISQ.DIST.RT function syntax has the following arguments:

- **X** Required. The value at which you want to evaluate the distribution.
- **Deg_freedom** Required. The number of degrees of freedom.

Remarks

- If either argument is nonnumeric, CHISQ.DIST.RT function returns the #VALUE! error value.
- If any argument is nonnumeric, CHISQ.DIST.RT function returns the #VALUE! error value.
- If deg_freedom is not an integer, it is truncated.
- If deg_freedom < 1 or deg_freedom > 10¹⁰, CHISQ.DIST.RT returns the #NUM! error value.

CHISQ.INV function

Description

Returns the inverse of the left-tailed probability of the chi-squared distribution.

The chi-squared distribution is commonly used to study variation in the percentage of something across samples, such as the fraction of the day people spend watching television.

Syntax

```
CHISQ.INV(probability,deg_freedom)
```

The CHISQ.INV function syntax has the following arguments:

- **Probability** Required. A probability associated with the chi-squared distribution.
- **Deg_freedom** Required. The number of degrees of freedom.

Remarks

- If argument is nonnumeric, CHISQ.INV returns the #VALUE! error value.
- If probability < 0 or probability > 1, CHISQ.INV returns the #NUM! error value.
- If deg_freedom is not an integer, it is truncated.
- If deg_freedom < 1 or deg_freedom > 10¹⁰, CHISQ.INV returns the #NUM! error value.

CHISQ.INV.RT function

Description

Returns the inverse of the right-tailed probability of the chi-squared distribution.

If $\text{probability} = \text{CHISQ.DIST.RT}(x, \dots)$, then $\text{CHISQ.INV.RT}(\text{probability}, \dots) = x$. Use this function to compare observed results with expected ones in order to decide whether your original hypothesis is valid.

Syntax

`CHISQ.INV.RT(probability,deg_freedom)`

The CHISQ.INV.RT function syntax has the following arguments:

- **Probability** Required. A probability associated with the chi-squared distribution.
- **Deg_freedom** Required. The number of degrees of freedom.

Remarks

- If either argument is nonnumeric, CHISQ.INV.RT returns the #VALUE! error value.
- If $\text{probability} < 0$ or $\text{probability} > 1$, CHISQ.INV.RT returns the #NUM! error value.
- If **deg_freedom** is not an integer, it is truncated.
- If $\text{deg_freedom} < 1$, CHISQ.INV.RT returns the #NUM! error value.

Given a value for **probability**, CHISQ.INV.RT seeks that value x such that $\text{CHISQ.DIST.RT}(x, \text{deg_freedom}) = \text{probability}$. Thus, precision of CHISQ.INV.RT depends on precision of CHISQ.DIST.RT. CHISQ.INV.RT uses an iterative search technique. If the search has not converged after 64 iterations, the function returns the #N/A error value.

CHISQ.TEST function

Description

Returns the test for independence. CHISQ.TEST returns the value from the chi-squared (χ^2) distribution for the statistic and the appropriate degrees of freedom. You can use χ^2 tests to determine whether hypothesized results are verified by an experiment.

Syntax

```
CHISQ.TEST(actual_range, expected_range)
```

The CHISQ.TEST function syntax has the following arguments:

- **Actual_range** Required. The range of data that contains observations to test against expected values.
- **Expected_range** Required. The range of data that contains the ratio of the product of row totals and column totals to the grand total.

Remarks

- If actual_range and expected_range have a different number of data points, CHISQ.TEST returns the #N/A error value.
- The χ^2 test first calculates a χ^2 statistic using the formula:

$$\chi^2 = \sum_{i=1}^r \sum_{j=1}^c \frac{(A_{ij} - E_{ij})^2}{E_{ij}}$$

where:

A_{ij} = actual frequency in the i-th row, j-th column

E_{ij} = expected frequency in the i-th row, j-th column

r = number of rows

c = number of columns

- A low value of χ^2 is an indicator of independence. As can be seen from the formula, χ^2 is always positive or 0, and is 0 only if $A_{ij} = E_{ij}$ for every i, j .
- CHISQ.TEST returns the probability that a value of the χ^2 statistic at least as high as the value calculated by the above formula could have happened by chance under the assumption of independence. In computing this probability, CHISQ.TEST uses the χ^2 distribution with an appropriate number of degrees of freedom, df . If $r > 1$ and $c > 1$, then $df = (r - 1)(c - 1)$. If $r = 1$ and $c > 1$, then $df = c - 1$ or if $r > 1$ and $c = 1$, then $df = r - 1$. $r = c = 1$ is not allowed and #N/A is returned.
- Use of CHISQ.TEST is most appropriate when E_{ij} 's are not too small. Some statisticians suggest that each E_{ij} should be greater than or equal to 5.

CONFIDENCE.NORM function

Description

Returns the confidence interval for a population mean, using a normal distribution.

The confidence interval is a range of values. Your sample mean, x , is at the center of this range and the range is $x \pm \text{CONFIDENCE.NORM}$. For example, if x is the sample mean of delivery times for products ordered through the mail, $x \pm \text{CONFIDENCE.NORM}$ is a range of population means. For any population mean, μ_0 , in this range, the probability of obtaining a sample mean further from μ_0 than x is greater than alpha; for any population mean, μ_0 , not in this range, the probability of obtaining a sample mean further from μ_0 than x is less than alpha. In other words, assume that we use x , `standard_dev`, and `size` to construct a two-tailed test at significance level alpha of the hypothesis that the population mean is μ_0 . Then we will not reject that hypothesis if μ_0 is in the confidence interval and will reject that hypothesis if μ_0 is not in the confidence interval. The confidence interval does not allow us to infer that there is probability $1 - \text{alpha}$ that our next package will take a delivery time that is in the confidence interval.

Syntax

```
CONFIDENCE.NORM(alpha, standard_dev, size)
```

The CONFIDENCE.NORM function syntax has the following **arguments**:

- **Alpha** Required. The significance level used to compute the confidence level. The confidence level equals $100 \times (1 - \text{alpha})\%$, or in other words, an alpha of 0.05 indicates a 95 percent confidence level.
- **Standard_dev** Required. The population standard deviation for the data range and is assumed to be known.
- **Size** Required. The sample size.

Remarks

- If any argument is nonnumeric, CONFIDENCE.NORM returns the #VALUE! error value.
- If $\text{alpha} \leq 0$ or $\text{alpha} \geq 1$, CONFIDENCE.NORM returns the #NUM! error value.

- If standard_dev ≤ 0, CONFIDENCE.NORM returns the #NUM! error value.
- If size is not an integer, it is truncated.
- If size < 1, CONFIDENCE.NORM returns the #NUM! error value.
- If we assume alpha equals 0.05, we need to calculate the area under the standard normal curve that equals (1 - alpha), or 95 percent. This value is ± 1.96. The confidence interval is therefore:

$$\bar{x} \pm 1.96 \left(\frac{\sigma}{\sqrt{n}} \right)$$

CONFIDENCE.T function

Description

Returns the confidence interval for a population mean, using a Student's t distribution.

Syntax

```
CONFIDENCE.T(alpha, standard_dev, size)
```

The CONFIDENCE.T function syntax has the following arguments:

- **Alpha** Required. The significance level used to compute the confidence level. The confidence level equals $100 \times (1 - \text{alpha})\%$, or in other words, an alpha of 0.05 indicates a 95 percent confidence level.
- **Standard_dev** Required. The population standard deviation for the data range and is assumed to be known.
- **Size** Required. The sample size.

Remarks

- If any argument is nonnumeric, CONFIDENCE.T returns the #VALUE! error value.
- If $\text{alpha} \leq 0$ or $\text{alpha} \geq 1$, CONFIDENCE.T returns the #NUM! error value.
- If $\text{standard_dev} \leq 0$, CONFIDENCE.T returns the #NUM! error value.
- If size is not an integer, it is truncated.
- If size equals 1, CONFIDENCE.T returns #DIV/0! error value.

CORREL function

Description

Returns the correlation coefficient of the array1 and array2 cell ranges. Use the correlation coefficient to determine the relationship between two properties. For example, you can examine the relationship between a location's average temperature and the use of air conditioners.

Syntax

```
CORREL(array1, array2)
```

The CORREL function syntax has the following arguments:

- **Array1** Required. A cell range of values.
- **Array2** Required. A second cell range of values.

Remarks

- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array1 and array2 have a different number of data points, CORREL returns the #N/A error value.
- If either array1 or array2 is empty, or if s (the standard deviation) of their values equals zero, CORREL returns the #DIV/0! error value.
- The equation for the correlation coefficient is:

$$\text{Correl}(X, Y) = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

where x and y are the sample means AVERAGE(array1) and AVERAGE(array2).

COUNT function

Description

The **COUNT** function counts the number of cells that contain numbers, and counts numbers within the list of arguments. Use the **COUNT** function to get the number of entries in a number field that is in a range or array of numbers. For example, you can enter the following formula to count the numbers in the range A1:A20:

```
=COUNT (A1 :A20)
```

In this example, if five of the cells in the range contain numbers, the result is **5**.

Syntax

```
COUNT (value1, [value2], ...)
```

The COUNT function syntax has the following arguments:

- **value1** Required. The first item, cell reference, or range within which you want to count numbers.
- **value2, ...** Optional. Up to 255 additional items, cell references, or ranges within which you want to count numbers.

NOTE The arguments can contain or refer to a variety of different types of data, but only numbers are counted.

Remarks

- Arguments that are numbers, dates, or a text representation of numbers (for example, a number enclosed in quotation marks, such as "1") are counted.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- Arguments that are error values or text that cannot be translated into numbers are not counted.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are not counted.

- If you want to count logical values, text, or error values, use the **COUNTA** function.
- If you want to count only numbers that meet certain criteria, use the **COUNTIF** function or the **COUNTIFS** function.

COUNTA function

Description

The **COUNTA** function counts the number of cells that are not empty in a **range**.

Syntax

```
COUNTA(value1, [value2], ...)
```

The COUNTA function syntax has the following arguments:

- **value1** Required. The first argument representing the values that you want to count.
- **value2, ...** Optional. Additional arguments representing the values that you want to count, up to a maximum of 255 arguments.

Remarks

- The **COUNTA** function counts cells containing any type of information, including error values and empty text (""). For example, if the range contains a formula that returns an empty string, the **COUNTA** function counts that value. The **COUNTA** function does not count empty cells.
- If you do not need to count logical values, text, or error values (in other words, if you want to count only cells that contain numbers), use the **COUNT** function.
- If you want to count only cells that meet certain criteria, use the **COUNTIF** function or the **COUNTIFS** function.

COUNTBLANK function

Description

Counts empty cells in a specified range of cells.

Syntax

```
COUNTBLANK (range)
```

The COUNTBLANK function syntax has the following arguments:

- **Range** Required. The range from which you want to count the blank cells.

Remark

Cells with formulas that return "" (empty text) are also counted. Cells with zero values are not counted.

COUNTIF function

Description

The **COUNTIF** function counts the number of cells within a range that meet a single criterion that you specify. For example, you can count all the cells that start with a certain letter, or you can count all the cells that contain a number that is larger or smaller than a number you specify. For example, suppose you have a worksheet that contains a list of tasks in column A, and the first name of the person assigned to each task in column B. You can use the **COUNTIF** function to count how many times a person's name appears in column B and, in that way, determine how many tasks are assigned to that person. For example:

```
=COUNTIF(B2:B25, "Nancy")
```

Syntax


```
COUNTIF(range, criteria)
```

The COUNTIF function syntax has the following arguments:

- **range** Required. One or more cells to count, including numbers or names, arrays, or references that contain numbers. Blank and text values are ignored.
- **criteria** Required. A number, expression, cell reference, or text string that defines which cells will be counted. For example, criteria can be expressed as 32, ">32", B4, "apples", or "32".

NOTES

- You can use the wildcard characters — the question mark (?) and the asterisk (*) — in criteria. A question mark matches any single character, and an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.
- Criteria are case insensitive; for example, the string "apples" and the string "APPLES" will match the same cells.

NOTE To view a number as a percentage, select the cell and then, on the **Home** tab, in the **Number** group, click **Percentage Style** .

COUNTIFS function

Description

Applies criteria to cells across multiple ranges and counts the number of times all criteria are met.

Syntax

```
COUNTIFS(criteria_range1, criteria1, [criteria_range2, criteria2]...)
```

The COUNTIFS function syntax has the following arguments:

- **criteria_range1** Required. The first range in which to evaluate the associated criteria.
- **criteria1** Required. The criteria in the form of a number, expression, cell reference, or text that define which cells will be counted. For example, criteria can be expressed as 32, ">32", B4, "apples", or "32".
- **criteria_range2, criteria2, ...** Optional. Additional ranges and their associated criteria. Up to 127 range/criteria pairs are allowed.

IMPORTANT Each additional range must have the same number of rows and columns as the **criteria_range1** argument. The ranges do not have to be adjacent to each other.

Remarks

- Each range's criteria is applied one cell at a time. If all of the first cells meet their associated criteria, the count increases by 1. If all of the second cells meet their associated criteria, the count increases by 1 again, and so on until all of the cells are evaluated.
- If the criteria argument is a reference to an empty cell, the **COUNTIFS** function treats the empty cell as a 0 value.
- You can use the wildcard characters— the question mark (?) and asterisk (*) — in criteria. A question mark matches any single character, and an asterisk matches any sequence of characters. If you want to find an actual question mark or asterisk, type a tilde (~) before the character.

COVARIANCE.P function

Description

Returns population covariance, the average of the products of deviations for each data point pair in two data sets.

Use covariance to determine the relationship between two data sets. For example, you can examine whether greater income accompanies greater levels of education.

Syntax

```
COVARIANCE.P(array1, array2)
```

The COVARIANCE.P function syntax has the following arguments:

- **Array1** Required. The first cell range of integers.
- **Array2** Required. The second cell range of integers.

Remarks

- The arguments must either be numbers or be names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array1 and array2 have different numbers of data points, COVARIANCE.P returns the #N/A error value.
- If either array1 or array2 is empty, COVARIANCE.P returns the #DIV/0! error value.
- The covariance is:

$$\text{Cov}(X, Y) = \frac{\sum (x - \bar{x})(y - \bar{y})}{n}$$

where x and y are the sample means $\text{AVERAGE}(\text{array1})$ and $\text{AVERAGE}(\text{array2})$, and n is the sample size.

COVARIANCE.S function

Description

Returns the sample covariance, the average of the products of deviations for each data point pair in two data sets.

Syntax

```
COVARIANCE.S(array1,array2)
```

The COVARIANCE.S function syntax has the following arguments:

- **Array1** Required. The first cell range of integers.
- **Array2** Required. The second cell range of integers.

Remarks

- The arguments must either be numbers or be names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array1 and array2 have different numbers of data points, COVARIANCE.S returns the #N/A error value.
- If either array1 or array2 is empty or contains only 1 data point each, COVARIANCE.S returns the #DIV/0! error value.

DEVSQ function

Description

Returns the sum of squares of deviations of data points from their sample mean.

Syntax

```
DEVSQ(number1, [number2], ...)
```

The DEVSQ function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want to calculate the sum of squared deviations. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- The equation for the sum of squared deviations is:

$$\text{DEVSQ} = \sum (x - \bar{x})^2$$

EXPON.DIST function

Description

Returns the exponential distribution. Use EXPON.DIST to model the time between events, such as how long an automated bank teller takes to deliver cash. For example, you can use EXPON.DIST to determine the probability that the process takes at most 1 minute.

Syntax

```
EXPON.DIST(x, lambda, cumulative)
```

The EXPON.DIST function syntax has the following arguments:

- **X** Required. The value of the function.
- **Lambda** Required. The parameter value.
- **Cumulative** Required. A logical value that indicates which form of the exponential function to provide. If cumulative is TRUE, EXPON.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- If x or lambda is nonnumeric, EXPON.DIST returns the #VALUE! error value.
- If $x < 0$, EXPON.DIST returns the #NUM! error value.
- If $\lambda \leq 0$, EXPON.DIST returns the #NUM! error value.
- The equation for the probability density function is:

$$f(x; \lambda) = \lambda e^{-\lambda x}$$

- The equation for the cumulative distribution function is:

$$F(x; \lambda) = 1 - e^{-\lambda x}$$

F.DIST function

Description

Returns the F probability distribution. You can use this function to determine whether two data sets have different degrees of diversity. For example, you can examine the test scores of men and women entering high school and determine if the variability in the females is different from that found in the males.

Syntax

```
F.DIST(x,deg_freedom1,deg_freedom2,cumulative)
```

The F.DIST function syntax has the following arguments:

- **X** Required. The value at which to evaluate the function.
- **Deg_freedom1** Required. The numerator degrees of freedom.
- **Deg_freedom2** Required. The denominator degrees of freedom.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, F.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- If any argument is nonnumeric, F.DIST returns the #VALUE! error value.
- If x is negative, F.DIST returns the #NUM! error value.
- If deg_freedom1 or deg_freedom2 is not an integer, it is truncated.
- If deg_freedom1 < 1, F.DIST returns the #NUM! error value.
- If deg_freedom2 < 1, F.DIST returns the #NUM! error value.

F.DIST.RT function

Description

Returns the (right-tailed) F probability distribution (degree of diversity) for two data sets.

You can use this function to determine whether two data sets have different degrees of diversity. For example, you can examine the test scores of men and women entering high school and determine if the variability in the females is different from that found in the males.

Syntax

```
F.DIST.RT(x,deg_freedom1,deg_freedom2)
```

The F.DIST.RT function syntax has the following arguments:

- **X** Required. The value at which to evaluate the function.
- **Deg_freedom1** Required. The numerator degrees of freedom.
- **Deg_freedom2** Required. The denominator degrees of freedom.

Remarks

- If any argument is nonnumeric, F.DIST.RT returns the #VALUE! error value.
- If x is negative, F.DIST.RT returns the #NUM! error value.
- If deg_freedom1 or deg_freedom2 is not an integer, it is truncated.
- If deg_freedom1 < 1 F.DIST.RT returns the #NUM! error value.
- If deg_freedom2 < 1 F.DIST.RT returns the #NUM! error value.
- F.DIST.RT is calculated as $F.DIST.RT = P(F > x)$, where F is a random variable that has an F distribution with deg_freedom1 and deg_freedom2 degrees of freedom.

F.INV function

Description

Returns the inverse of the F probability distribution. If $p = F.DIST(x, \dots)$, then $F.INV(p, \dots) = x$. The F distribution can be used in an F-test that compares the degree of variability in two data sets. For example, you can analyze income distributions in the United States and Canada to determine whether the two countries have a similar degree of income diversity.

Syntax

```
F.INV(probability, deg_freedom1, deg_freedom2)
```

The F.INV function syntax has the following arguments:

- **Probability** Required. A probability associated with the F cumulative distribution.
- **Deg_freedom1** Required. The numerator degrees of freedom.
- **Deg_freedom2** Required. The denominator degrees of freedom.

Remarks

- If any argument is nonnumeric, F.INV returns the #VALUE! error value.
- If probability < 0 or probability > 1, F.INV returns the #NUM! error value.
- If deg_freedom1 or deg_freedom2 is not an integer, it is truncated.
- If deg_freedom1 < 1, or deg_freedom2 < 1, F.INV returns the #NUM! error value.

F.INV.RT function

Description

Returns the inverse of the (right-tailed) F probability distribution. If $p = F.DIST.RT(x, \dots)$, then $F.INV.RT(p, \dots) = x$.

The F distribution can be used in an F-test that compares the degree of variability in two data sets. For example, you can analyze income distributions in the United States and Canada to determine whether the two countries have a similar degree of income diversity.

Syntax

```
F.INV.RT(probability, deg_freedom1, deg_freedom2)
```

The F.INV.RT function syntax has the following arguments:

- **Probability** Required. A probability associated with the F cumulative distribution.
- **Deg_freedom1** Required. The numerator degrees of freedom.
- **Deg_freedom2** Required. The denominator degrees of freedom.

Remarks

- If any argument is nonnumeric, F.INV.RT returns the #VALUE! error value.
- If probability < 0 or probability > 1, F.INV.RT returns the #NUM! error value.
- If deg_freedom1 or deg_freedom2 is not an integer, it is truncated.
- If deg_freedom1 < 1, or deg_freedom2 < 1, F.INV.RT returns the #NUM! error value.
- If deg_freedom2 < 1 or deg_freedom2 $\geq 10^{10}$, F.INV.RT returns the #NUM! error value.

F.INV.RT can be used to return critical values from the F distribution. For example, the output of an ANOVA calculation often includes data for the F statistic, F probability, and F critical value at the 0.05 significance level. To return the critical value of F, use the significance level as the probability argument to F.INV.RT.

Given a value for probability, F.INV.RT seeks that value x such that $F.DIST.RT(x, \text{deg_freedom1}, \text{deg_freedom2}) = \text{probability}$. Thus, precision of F.INV.RT depends on precision of F.DIST.RT. F.INV.RT uses an iterative search technique. If the search has not converged after 64 iterations, the function returns the #N/A error value.

F.TEST function

Description

Returns the result of an F-test, the two-tailed probability that the variances in array1 and array2 are not significantly different.

Use this function to determine whether two samples have different variances. For example, given test scores from public and private schools, you can test whether these schools have different levels of test score diversity.

Syntax

```
F.TEST(array1, array2)
```

The F.TEST function syntax has the following arguments:

- **Array1** Required. The first array or range of data.
- **Array2** Required. The second array or range of data.

Remarks

- The arguments must be either numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If the number of data points in array1 or array2 is less than 2, or if the variance of array1 or array2 is zero, F.TEST returns the #DIV/0! error value.

FISHER function

Description

Returns the Fisher transformation at x. This transformation produces a function that is normally distributed rather than skewed. Use this function to perform hypothesis testing on the correlation coefficient.

Syntax

```
FISHER(x)
```

The FISHER function syntax has the following arguments:

- **X** Required. A numeric value for which you want the transformation.

Remarks

- If x is nonnumeric, FISHER returns the #VALUE! error value.
- If $x \leq -1$ or if $x \geq 1$, FISHER returns the #NUM! error value.
- The equation for the Fisher transformation is:

$$z' = \frac{1}{2} \ln \left(\frac{1+x}{1-x} \right)$$

FISHERINV function

Description

Returns the inverse of the Fisher transformation. Use this transformation when analyzing correlations between ranges or arrays of data. If $y = \text{FISHER}(x)$, then $\text{FISHERINV}(y) = x$.

Syntax

```
FISHERINV (y)
```

The FISHERINV function syntax has the following arguments:

- **Y** Required. The value for which you want to perform the inverse of the transformation.

Remarks

- If y is nonnumeric, FISHERINV returns the #VALUE! error value.
- The equation for the inverse of the Fisher transformation is:

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

FORECAST function

Description

Calculates, or predicts, a future value by using existing values. The predicted value is a y-value for a given x-value. The known values are existing x-values and y-values, and the new value is predicted by using linear regression. You can use this function to predict future sales, inventory requirements, or consumer trends.

Syntax

```
FORECAST(x, known_y's, known_x's)
```

The FORECAST function syntax has the following arguments:

- **X** Required. The data point for which you want to predict a value.
- **Known_y's** Required. The dependent array or range of data.
- **Known_x's** Required. The independent array or range of data.

Remarks

- If x is nonnumeric, FORECAST returns the #VALUE! error value.
- If known_y's and known_x's are empty or contain a different number of data points, FORECAST returns the #N/A error value.
- If the variance of known_x's equals zero, then FORECAST returns the #DIV/0! error value.
- The equation for FORECAST is $a+bx$, where:

$$a = \bar{y} - b\bar{x}$$

and:

$$b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

and where x and y are the sample means AVERAGE(known_x's) and AVERAGE(known y's).

FREQUENCY function

Description

Calculates how often values occur within a range of values, and then returns a vertical array of numbers. For example, use FREQUENCY to count the number of test scores that fall within ranges of scores. Because FREQUENCY returns an array, it must be entered as an array formula.

Syntax

```
FREQUENCY(data_array, bins_array)
```

The FREQUENCY function syntax has the following arguments:

- **Data_array** Required. An array of or reference to a set of values for which you want to count frequencies. If data_array contains no values, FREQUENCY returns an array of zeros.
- **Bins_array** Required. An array of or reference to intervals into which you want to group the values in data_array. If bins_array contains no values, FREQUENCY returns the number of elements in data_array.

Remarks

- FREQUENCY is entered as an array formula after you select a range of adjacent cells into which you want the returned distribution to appear.
- The number of elements in the returned array is one more than the number of elements in bins_array. The extra element in the returned array returns the count of any values above the highest interval. For example, when counting three ranges of values (intervals) that are entered into three cells, be sure to enter FREQUENCY into four cells for the results. The extra cell returns the number of values in data_array that are greater than the third interval value.
- FREQUENCY ignores blank cells and text.
- Formulas that return arrays must be entered as array formulas.

GAMMA.DIST function

Description

Returns the gamma distribution. You can use this function to study variables that may have a skewed distribution. The gamma distribution is commonly used in queuing analysis.

Syntax

```
GAMMA.DIST(x, alpha, beta, cumulative)
```

The GAMMA.DIST function syntax has the following arguments:

- **X** Required. The value at which you want to evaluate the distribution.
- **Alpha** Required. A parameter to the distribution.
- **Beta** Required. A parameter to the distribution. If beta = 1, GAMMA.DIST returns the standard gamma distribution.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, GAMMA.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- If x, alpha, or beta is nonnumeric, GAMMA.DIST returns the #VALUE! error value.
- If x < 0, GAMMA.DIST returns the #NUM! error value.
- If alpha ≤ 0 or if beta ≤ 0, GAMMA.DIST returns the #NUM! error value.
- The equation for the gamma probability density function is:

$$f(x; \alpha, \beta) = \frac{1}{\beta^\alpha \Gamma(\alpha)} x^{\alpha-1} e^{-\frac{x}{\beta}}$$

The standard gamma probability density function is:

$$f(x; \alpha) = \frac{x^{\alpha-1} e^{-x}}{\Gamma(\alpha)}$$

- When alpha = 1, GAMMA.DIST returns the exponential distribution with:

$$\lambda = \frac{1}{\theta}$$

- For a positive integer n, when alpha = n/2, beta = 2, and cumulative = TRUE, GAMMA.DIST returns (1 - CHISQ.DIST.RT(x)) with n degrees of freedom.
- When alpha is a positive integer, GAMMA.DIST is also known as the Erlang distribution.

GAMMA.INV function

Description

Returns the inverse of the gamma cumulative distribution. If $p = \text{GAMMA.DIST}(x, \dots)$, then $\text{GAMMA.INV}(p, \dots) = x$.

You can use this function to study a variable whose distribution may be skewed.

Syntax

```
GAMMA.INV(probability, alpha, beta)
```

The GAMMA.INV function syntax has the following arguments:

- **Probability** Required. The probability associated with the gamma distribution.
- **Alpha** Required. A parameter to the distribution.
- **Beta** Required. A parameter to the distribution. If beta = 1, GAMMA.INV returns the standard gamma distribution.

Remarks

- If any argument is text, GAMMA.INV returns the #VALUE! error value.
- If probability < 0 or probability > 1, GAMMA.INV returns the #NUM! error value.
- If alpha ≤ 0 or if beta ≤ 0, GAMMA.INV returns the #NUM! error value.

Given a value for probability, GAMMA.INV seeks that value x such that $\text{GAMMA.DIST}(x, \text{alpha}, \text{beta}, \text{TRUE}) = \text{probability}$. Thus, precision of GAMMA.INV depends on precision of GAMMA.DIST. GAMMA.INV uses an iterative search technique. If the search has not converged after 64 iterations, the function returns the #N/A error value.

GAMMALN function

Description

Returns the natural logarithm of the gamma function, $\Gamma(x)$.

Syntax

`GAMMALN (x)`

The GAMMALN function syntax has the following arguments:

- **X** Required. The value for which you want to calculate GAMMALN.

Remarks

- If x is nonnumeric, GAMMALN returns the #VALUE! error value.
- If $x \leq 0$, GAMMALN returns the #NUM! error value.
- The number e raised to the GAMMALN(i) power, where i is an integer, returns the same result as (i - 1)!.
- GAMMALN is calculated as follows:

$$GAMMALN = LN(\Gamma(x))$$

where:

$$\Gamma(x) = \int_0^{\infty} e^{-u} u^{x-1} du$$

GAMMALN.PRECISE function

Description

Returns the natural logarithm of the gamma function, $\Gamma(x)$.

Syntax

```
GAMMALN.PRECISE (x)
```

The GAMMALN.PRECISE function syntax has the following arguments:

- **X** Required. The value for which you want to calculate GAMMALN.PRECISE.

Remarks

- If x is nonnumeric, GAMMALN.PRECISE returns the #VALUE! error value.
- If $x \leq 0$, GAMMALN.PRECISE returns the #NUM! error value.
- The number e raised to the GAMMALN.PRECISE(i) power, where i is an integer, returns the same result as (i - 1)!.
- GAMMALN.PRECISE is calculated as follows:

$$\text{GAMMALN.PRECISE} = \text{LN}(\Gamma(x))$$

where:

$$\Gamma(x) = \int_0^{\infty} e^{-u} u^{x-1} du$$

GEOMEAN function

Description

Returns the geometric mean of an array or range of positive data. For example, you can use GEOMEAN to calculate average growth rate given compound interest with variable rates.

Syntax

```
GEOMEAN(number1, [number2], ...)
```

The GEOMEAN function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want to calculate the mean. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If any data point ≤ 0 , GEOMEAN returns the #NUM! error value.
- The equation for the geometric mean is:

$$GM_{\bar{y}} = \sqrt[n]{y_1 y_2 y_3 \dots y_n}$$

GROWTH function

Description

Calculates predicted exponential growth by using existing data. GROWTH returns the y-values for a series of new x-values that you specify by using existing x-values and y-values. You can also use the GROWTH worksheet function to fit an exponential curve to existing x-values and y-values.

Syntax

```
GROWTH(known_y's, [known_x's], [new_x's], [const])
```

The GROWTH function syntax has the following arguments:

- **Known_y's** Required. The set of y-values you already know in the relationship $y = b \cdot m^x$.
 - If the array known_y's is in a single column, then each column of known_x's is interpreted as a separate variable.
 - If the array known_y's is in a single row, then each row of known_x's is interpreted as a separate variable.
 - If any of the numbers in known_y's is 0 or negative, GROWTH returns the #NUM! error value.
- **Known_x's** Optional. An optional set of x-values that you may already know in the relationship $y = b \cdot m^x$.
 - The array known_x's can include one or more sets of variables. If only one variable is used, known_y's and known_x's can be ranges of any shape, as long as they have equal dimensions. If more than one variable is used, known_y's must be a vector (that is, a range with a height of one row or a width of one column).
 - If known_x's is omitted, it is assumed to be the array {1,2,3,...} that is the same size as known_y's.
- **New_x's** Optional. Are new x-values for which you want GROWTH to return corresponding y-values.

- New_x's must include a column (or row) for each independent variable, just as known_x's does. So, if known_y's is in a single column, known_x's and new_x's must have the same number of columns. If known_y's is in a single row, known_x's and new_x's must have the same number of rows.
- If new_x's is omitted, it is assumed to be the same as known_x's.
- If both known_x's and new_x's are omitted, they are assumed to be the array {1,2,3,...} that is the same size as known_y's.
- **Const** Optional. A logical value specifying whether to force the constant b to equal 1.
 - If const is TRUE or omitted, b is calculated normally.
 - If const is FALSE, b is set equal to 1 and the m-values are adjusted so that $y = m^x$.

Remarks

- Formulas that return arrays must be entered as array formulas after selecting the correct number of cells.
- When entering an array constant for an argument such as known_x's, use commas to separate values in the same row and semicolons to separate rows.

HARMEAN function

Description

Returns the harmonic mean of a data set. The harmonic mean is the reciprocal of the arithmetic mean of reciprocals.

Syntax

```
HARMEAN(number1, [number2], ...)
```

The HARMEAN function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want to calculate the mean. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- The harmonic mean is always less than the geometric mean, which is always less than the arithmetic mean.
- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If any data point ≤ 0 , HARMEAN returns the #NUM! error value.
- The equation for the harmonic mean is:

$$\frac{1}{H_y} = \frac{1}{n} \sum \frac{1}{Y_i}$$

HYPGEOM.DIST function

HYPGEOM.DIST returns the probability of a given number of sample successes, given the sample size, population successes, and population size. Use HYPGEOM.DIST for problems with a finite population, where each observation is either a success or a failure, and where each subset of a given size is chosen with equal likelihood.

Syntax

```
HYPGEOM.DIST(sample_s,number_sample,population_s,number_pop,cumulative)
```

The HYPGEOM.DIST function syntax has the following arguments:

- **Sample_s** Required. The number of successes in the sample.
- **Number_sample** Required. The size of the sample.
- **Population_s** Required. The number of successes in the population.
- **Number_pop** Required. The population size.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, then HYPGEOM.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.

Remarks

- All arguments are truncated to integers.
- If any argument is nonnumeric, HYPGEOM.DIST returns the #VALUE! error value.
- If $sample_s < 0$ or $sample_s$ is greater than the lesser of $number_sample$ or $population_s$, HYPGEOM.DIST returns the #NUM! error value.
- If $sample_s$ is less than the larger of 0 or $(number_sample - number_population + population_s)$, HYPGEOM.DIST returns the #NUM! error value.
- If $number_sample \leq 0$ or $number_sample > number_population$, HYPGEOM.DIST returns the #NUM! error value.
- If $population_s \leq 0$ or $population_s > number_population$, HYPGEOM.DIST returns the #NUM! error value.

- If $\text{number_pop} \leq 0$, HYPGEOM.DIST returns the #NUM! error value.
- The equation for the hypergeometric distribution is:

$$P(X = x) = h(x; n, M, N) = \frac{\binom{M}{x} \binom{N - M}{n - x}}{\binom{N}{n}}$$

where:

x = sample_s

n = number_sample

M = population_s

N = number_pop

HYPGEOM.DIST is used in sampling without replacement from a finite population.

INTERCEPT function

Description

Calculates the point at which a line will intersect the y-axis by using existing x-values and y-values. The intercept point is based on a best-fit regression line plotted through the known x-values and known y-values. Use the INTERCEPT function when you want to determine the value of the dependent variable when the independent variable is 0 (zero). For example, you can use the INTERCEPT function to predict a metal's electrical resistance at 0°C when your data points were taken at room temperature and higher.

Syntax

```
INTERCEPT (known_y's, known_x's)
```

The INTERCEPT function syntax has the following arguments:

- **Known_y's** Required. The dependent set of observations or data.
- **Known_x's** Required. The independent set of observations or data.

Remarks

- The arguments should be either numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If known_y's and known_x's contain a different number of data points or contain no data points, INTERCEPT returns the #N/A error value.
- The equation for the intercept of the regression line, a, is:

$$a = \bar{y} - b\bar{x}$$

where the slope, b, is calculated as:

$$b = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

and where x and y are the sample means $AVERAGE(\text{known_x's})$ and $AVERAGE(\text{known_y's})$.

- ↓ The underlying algorithm used in the INTERCEPT and SLOPE functions is different than the underlying algorithm used in the LINEST function. The difference between these algorithms can lead to different results when data is undetermined and collinear. For example, if the data points of the known_y's argument are 0 and the data points of the known_x's argument are 1:
 - INTERCEPT and SLOPE return a #DIV/0! error. The INTERCEPT and SLOPE algorithm is designed to look for one and only one answer, and in this case there can be more than one answer.
 - LINEST returns a value of 0. The LINEST algorithm is designed to return reasonable results for collinear data, and in this case at least one answer can be found.

KURT function

Description

Returns the kurtosis of a data set. Kurtosis characterizes the relative peakedness or flatness of a distribution compared with the normal distribution. Positive kurtosis indicates a relatively peaked distribution. Negative kurtosis indicates a relatively flat distribution.

Syntax

```
KURT(number1, [number2], ...)
```

The KURT function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want to calculate kurtosis. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If there are fewer than four data points, or if the standard deviation of the sample equals zero, KURT returns the #DIV/0! error value.
- Kurtosis is defined as:

$$\left\{ \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum \left(\frac{x_j - \bar{x}}{s} \right)^4 \right\} - \frac{3(n-1)^2}{(n-2)(n-3)}$$

where s is the sample standard deviation.

LARGE function

Description

Returns the k-th largest value in a data set. You can use this function to select a value based on its relative standing. For example, you can use LARGE to return the highest, runner-up, or third-place score.

Syntax

```
LARGE(array, k)
```

The LARGE function syntax has the following arguments:

- **Array** Required. The array or range of data for which you want to determine the k-th largest value.
- **K** Required. The position (from the largest) in the array or cell range of data to return.

Remarks

- If array is empty, LARGE returns the #NUM! error value.
- If $k \leq 0$ or if k is greater than the number of data points, LARGE returns the #NUM! error value.

If n is the number of data points in a range, then LARGE(array,1) returns the largest value, and LARGE(array,n) returns the smallest value.

LINEST function

Description

The **LINEST** function calculates the statistics for a line by using the "least squares" method to calculate a straight line that best fits your data, and then returns an array that describes the line. You can also combine **LINEST** with other functions to calculate the statistics for other types of models that are linear in the unknown parameters, including polynomial, logarithmic, exponential, and power series. Because this function returns an array of values, it must be entered as an array formula. Instructions follow the examples in this article.

The equation for the line is:

$$y = mx + b$$

–or–

$$y = m_1x_1 + m_2x_2 + \dots + b$$

if there are multiple ranges of x-values, where the dependent y-values are a function of the independent x-values. The m-values are coefficients corresponding to each x-value, and b is a constant value. Note that y, x, and m can be vectors. The array that the **LINEST** function returns is {m_n,m_{n-1},...,m₁,b}. **LINEST** can also return additional regression statistics.

Syntax

```
LINEST(known_y's, [known_x's], [const], [stats])
```

The LINEST function syntax has the following arguments:

- **known_y's** Required. The set of y-values that you already know in the relationship $y = mx + b$.
 - If the range of known_y's is in a single column, each column of known_x's is interpreted as a separate variable.
 - If the range of known_y's is contained in a single row, each row of known_x's is interpreted as a separate variable.

- **known_x's** Optional. A set of x-values that you may already know in the relationship $y = mx + b$.
 - The range of known_x's can include one or more sets of variables. If only one variable is used, known_y's and known_x's can be ranges of any shape, as long as they have equal dimensions. If more than one variable is used, known_y's must be a vector (that is, a range with a height of one row or a width of one column).
 - If known_x's is omitted, it is assumed to be the array {1,2,3,...} that is the same size as known_y's.
- **const** Optional. A logical value specifying whether to force the constant b to equal 0.
 - If const is TRUE or omitted, b is calculated normally.
 - If const is FALSE, b is set equal to 0 and the m-values are adjusted to fit $y = mx$.
- **stats** Optional. A logical value specifying whether to return additional regression statistics.
 - If stats is TRUE, LINEST returns the additional regression statistics; as a result, the returned array is {mn,mn-1,...,m1,b;sen,se1,...,se1,seb;r2,sey;F,df;ssreg,ssresid}.
 - If stats is FALSE or omitted, LINEST returns only the m-coefficients and the constant b.

The additional regression statistics are as follows.

Statistic	Description
se1,se2,...,sen	The standard error values for the coefficients m1,m2,...,mn.
seb	The standard error value for the constant b (seb = #N/A when const is FALSE).
r2	The coefficient of determination. Compares estimated and actual y-values, and ranges in value from 0 to 1. If it is 1, there is a perfect correlation in the sample — there is no difference between the estimated y-value and the actual y-value. At the other extreme, if the coefficient of determination is 0, the regression equation is not helpful in predicting a y-value. For information about how r2 is calculated, see "Remarks," later in this topic.
sey	The standard error for the y estimate.
F	The F statistic, or the F-observed value. Use the F statistic to determine whether the observed relationship between the dependent and independent variables occurs by chance.

df	The degrees of freedom. Use the degrees of freedom to help you find F-critical values in a statistical table. Compare the values you find in the table to the F statistic returned by LINEST to determine a confidence level for the model. For information about how df is calculated, see "Remarks," later in this topic. Example 4 shows use of F and df.
ssreg	The regression sum of squares.
ssresid	The residual sum of squares. For information about how ssreg and ssresid are calculated, see "Remarks," later in this topic.

The following illustration shows the order in which the additional regression statistics are returned.

	A	B	C	D	E	F
1	m_n	m_{n-1}	...	m_2	m_1	b
2	se_n	se_{n-1}	...	se_2	se_1	se_b
3	r^2	se_y				
4	F	df				
5	ssreg	ssresid				

Remarks

- You can describe any straight line with the slope and the y-intercept:

Slope (m):

To find the slope of a line, often written as m, take two points on the line, (x1,y1) and (x2,y2); the slope is equal to $(y_2 - y_1)/(x_2 - x_1)$.

Y-intercept (b):

The y-intercept of a line, often written as b, is the value of y at the point where the line crosses the y-axis.

The equation of a straight line is $y = mx + b$. Once you know the values of m and b, you can calculate any point on the line by plugging the y- or x-value into that equation. You can also use the **TREND** function.

- When you have only one independent x-variable, you can obtain the slope and y-intercept values directly by using the following formulas:

Slope:

`=INDEX(LINEST(known_y's,known_x's),1)`

Y-intercept:

=INDEX(LINEST(known_y's,known_x's),2)

- The accuracy of the line calculated by the **LINEST** function depends on the degree of scatter in your data. The more linear the data, the more accurate the **LINEST** model. **LINEST** uses the method of least squares for determining the best fit for the data. When you have only one independent x-variable, the calculations for m and b are based on the following formulas:

$$m = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sum (x - \bar{x})^2}$$

$$b = \bar{y} - m\bar{x}$$

where x and y are sample means; that is, x = **AVERAGE(known x's)** and y = **AVERAGE(known y's)**.

- The line- and curve-fitting functions **LINEST** and **LOGEST** can calculate the best straight line or exponential curve that fits your data. However, you have to decide which of the two results best fits your data. You can calculate **TREND(known_y's,known_x's)** for a straight line, or **GROWTH(known_y's, known_x's)** for an exponential curve. These functions, without the **new_x's** argument, return an array of y-values predicted along that line or curve at your actual data points. You can then compare the predicted values with the actual values. You may want to chart them both for a visual comparison.

LOGEST function

Description

In regression analysis, calculates an exponential curve that fits your data and returns an array of values that describes the curve. Because this function returns an array of values, it must be entered as an array formula.

The equation for the curve is:

$$y = b * m^x$$

or

$$y = (b * (m1^x1) * (m2^x2) * \dots)$$

if there are multiple x-values, where the dependent y-value is a function of the independent x-values. The m-values are bases corresponding to each exponent x-value, and b is a constant value. Note that y, x, and m can be vectors. The array that LOGEST returns is {m_n, m_{n-1}, ..., m₁, b}.

Syntax

```
LOGEST(known_y's, [known_x's], [const], [stats])
```

The LOGEST function syntax has the following arguments:

- **Known_y's** Required. The set of y-values you already know in the relationship $y = b * m^x$.
 - If the array known_y's is in a single column, then each column of known_x's is interpreted as a separate variable.
 - If the array known_y's is in a single row, then each row of known_x's is interpreted as a separate variable.
- **Known_x's** Optional. An optional set of x-values that you may already know in the relationship $y = b * m^x$.

- The array known_x's can include one or more sets of variables. If only one variable is used, known_y's and known_x's can be ranges of any shape, as long as they have equal dimensions. If more than one variable is used, known_y's must be a range of cells with a height of one row or a width of one column (which is also known as a vector).
 - If known_x's is omitted, it is assumed to be the array {1,2,3,...} that is the same size as known_y's.
- **Const** Optional. A logical value specifying whether to force the constant b to equal 1.
 - If const is TRUE or omitted, b is calculated normally.
 - If const is FALSE, b is set equal to 1, and the m-values are fitted to $y = m^x$.
- **Stats** Optional. A logical value specifying whether to return additional regression statistics.
 - If stats is TRUE, LOGEST returns the additional regression statistics, so the returned array is {mn,mn-1,...,m1,b;sen,sen-1,...,se1,seb;r 2,sey; F,df;ssreg,ssresid}.
 - If stats is FALSE or omitted, LOGEST returns only the m-coefficients and the constant b.

Remarks

- The more a plot of your data resembles an exponential curve, the better the calculated line will fit your data. Like LINEST, LOGEST returns an array of values that describes a relationship among the values, but LINEST fits a straight line to your data; LOGEST fits an exponential curve. For more information, see LINEST.
- When you have only one independent x-variable, you can obtain y-intercept (b) values directly by using the following formula:

Y-intercept (b):

`INDEX(LOGEST(known_y's,known_x's),2)`

- Formulas that return arrays must be entered as array formulas.

- When entering an array constant such as known_x's as an argument, use commas to separate values in the same row and semicolons to separate rows. Separator characters may be different depending on your locale setting in **Regional and Language Options** in **Control Panel**.
- You should note that the y-values predicted by the regression equation may not be valid if they are outside the range of y-values you used to determine the equation.

LOGNORM.DIST function

Description

Returns the lognormal distribution of x , where $\ln(x)$ is normally distributed with parameters Mean and Standard_dev.

Use this function to analyze data that has been logarithmically transformed.

Syntax

```
LOGNORM.DIST(x,mean,standard_dev,cumulative)
```

The LOGNORM.DIST function syntax has the following arguments:

- **X** Required. The value at which to evaluate the function.
- **Mean** Required. The mean of $\ln(x)$.
- **Standard_dev** Required. The standard deviation of $\ln(x)$.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, LOGNORM.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- If any argument is nonnumeric, LOGNORM.DIST returns the #VALUE! error value.
- If $x \leq 0$ or if $\text{standard_dev} \leq 0$, LOGNORM.DIST returns the #NUM! error value.
- The equation for the lognormal cumulative distribution function is:
- $\text{LOGNORM.DIST}(x,\mu,\sigma) = \text{NORM.S.DIST}(\ln(x)-\mu / \sigma)$

LOGNORM.INV function

Description

Returns the inverse of the lognormal cumulative distribution function of x , where $\ln(x)$ is normally distributed with parameters Mean and Standard_dev. If $p = \text{LOGNORM.DIST}(x, \dots)$ then $\text{LOGNORM.INV}(p, \dots) = x$.

Use the lognormal distribution to analyze logarithmically transformed data.

Syntax

```
LOGNORM.INV(probability, mean, standard_dev)
```

The LOGNORM.INV function syntax has the following arguments:

- **Probability** Required. A probability associated with the lognormal distribution.
- **Mean** Required. The mean of $\ln(x)$.
- **Standard_dev** Required. The standard deviation of $\ln(x)$.

Remarks

- If any argument is nonnumeric, LOGNORM.INV returns the #VALUE! error value.
- If probability ≤ 0 or probability ≥ 1 , LOGNORM.INV returns the #NUM! error value.
- If standard_dev ≤ 0 , LOGNORM.INV returns the #NUM! error value.

MAX function

Description

Returns the largest value in a set of values.

Syntax

```
MAX(number1, [number2], ...)
```

The MAX function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 numbers for which you want to find the maximum value.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are used. Empty cells, logical values, or text in the array or reference are ignored.
- If the arguments contain no numbers, MAX returns 0 (zero).
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MAXA function.

MAXA function

Description

Returns the largest value in a list of arguments.

MAXA is similar to MINA. For more information, see the examples for the MINA function.

Syntax

```
MAXA (value1, [value2], ...)
```

The MAXA function syntax has the following arguments:

- **Value1** Required. The first number argument for which you want to find the largest value.
- **Value2,...** Optional. Number arguments 2 to 255 for which you want to find the largest value.

Remarks

- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
- If the arguments contain no values, MAXA returns 0 (zero).
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the MAX function.

MEDIAN function

Description

Returns the median of the given numbers. The median is the number in the middle of a set of numbers.

Syntax

```
MEDIAN(number1, [number2], ...)
```

The MEDIAN function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 numbers for which you want the median.

Remarks

- If there is an even number of numbers in the set, then MEDIAN calculates the average of the two numbers in the middle. See the second formula in the example.
- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.

NOTE The MEDIAN function measures central tendency, which is the location of the center of a group of numbers in a statistical distribution. The three most common measures of central tendency are:

- **Average** which is the arithmetic mean, and is calculated by adding a group of numbers and then dividing by the count of those numbers. For example, the average of 2, 3, 3, 5, 7, and 10 is 30 divided by 6, which is 5.

- **Median** which is the middle number of a group of numbers; that is, half the numbers have values that are greater than the median, and half the numbers have values that are less than the median. For example, the median of 2, 3, 3, 5, 7, and 10 is 4.
- **Mode** which is the most frequently occurring number in a group of numbers. For example, the mode of 2, 3, 3, 5, 7, and 10 is 3.

For a symmetrical distribution of a group of numbers, these three measures of central tendency are all the same. For a skewed distribution of a group of numbers, they can be different.

MIN function

Description

Returns the smallest number in a set of values.

Syntax

```
MIN(number1, [number2], ...)
```

The MIN function syntax has the following arguments:

- **Number1, number2, ...** Number1 is optional, subsequent numbers are optional. 1 to 255 numbers for which you want to find the minimum value.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are used. Empty cells, logical values, or text in the array or reference are ignored.
- If the arguments contain no numbers, MIN returns 0.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the MINA function.

MINA function

Description

Returns the smallest value in the list of arguments.

Syntax

```
MINA(value1, [value2], ...)
```

The MINA function syntax has the following arguments:

- **Value1, value2, ...** Value1 is required, subsequent values are optional. 1 to 255 values for which you want to find the smallest value.

Remarks

- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If the arguments contain no values, MINA returns 0.
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the MIN function.

MODE.MULT function

Description

Returns a vertical array of the most frequently occurring, or repetitive values in an array or range of data. For horizontal arrays, use `TRANSPOSE(MODE.MULT(number1,number2,...))`.

This will return more than one result if there are multiple modes. Because this function returns an array of values, it must be entered as an array formula.

Syntax

```
MODE.MULT ( (number1, [number2], ... ) )
```

The `MODE.MULT` function syntax has the following arguments:

- **Number1** Required. The first number argument for which you want to calculate the mode.
- **Number2, ...** Optional. Number arguments 2 to 254 for which you want to calculate the mode. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If the data set contains no duplicate data points, `MODE.MULT` returns the `#N/A` error value.

MODE.SNGL function

Description

Returns the most frequently occurring, or repetitive, value in an array or range of data.

Syntax

```
MODE.SNGL(number1, [number2], ...)
```

The MODE.SNGL function syntax has the following arguments:

- **Number1** Required. The first argument for which you want to calculate the mode.
- **Number2, ...** Optional. Arguments 2 to 254 for which you want to calculate the mode. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If the data set contains no duplicate data points, MODE.SNGL returns the #N/A error value.

Note The MODE.SNGL function measures central tendency, which is the location of the center of a group of numbers in a statistical distribution. The three most common measures of central tendency are:

- **Average** which is the arithmetic mean, and is calculated by adding a group of numbers and then dividing by the count of those numbers. For example, the average of 2, 3, 3, 5, 7, and 10 is 30 divided by 6, which is 5.
- **Median** which is the middle number of a group of numbers; that is, half the numbers have values that are greater than the median, and half the numbers have values that are less than the median. For example, the median of 2, 3, 3, 5, 7, and 10 is 4.

- **Mode** which is the most frequently occurring number in a group of numbers. For example, the mode of 2, 3, 3, 5, 7, and 10 is 3.

For a symmetrical distribution of a group of numbers, these three measures of central tendency are all the same. For a skewed distribution of a group of numbers, they can be different.

NEGBINOM.DIST function

Description

Returns the negative binomial distribution, the probability that there will be **Number_f** failures before the **Number_s**-th success, with **Probability_s** probability of a success.

This function is similar to the binomial distribution, except that the number of successes is fixed, and the number of trials is variable. Like the binomial, trials are assumed to be independent.

For example, you need to find 10 people with excellent reflexes, and you know the probability that a candidate has these qualifications is 0.3. NEGBINOM.DIST calculates the probability that you will interview a certain number of unqualified candidates before finding all 10 qualified candidates.

Syntax

```
NEGBINOM.DIST(number_f,number_s,probability_s,cumulative)
```

The NEGBINOM.DIST function syntax has the following arguments:

- **Number_f** Required. The number of failures.
- **Number_s** Required. The threshold number of successes.
- **Probability_s** Required. The probability of a success.
- **Cumulative** Required. A logical value that determines the form of the function. If **cumulative** is TRUE, NEGBINOM.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- **Number_f** and **number_s** are truncated to integers.
- If any argument is nonnumeric, NEGBINOM.DIST returns the #VALUE! error value.
- If **probability_s** < 0 or if **probability** > 1, NEGBINOM.DIST returns the #NUM! error value.
- If **number_f** < 0 or **number_s** < 1, NEGBINOM.DIST returns the #NUM! error value.

- The equation for the negative binomial distribution is:

$$nb(x; r, p) = \binom{x+r-1}{r-1} p^r (1-p)^x$$

where:

x is number_f, r is number_s, and p is probability_s.

NORM.DIST function

Description

Returns the normal distribution for the specified mean and standard deviation. This function has a very wide range of applications in statistics, including hypothesis testing.

Syntax

```
NORM.DIST(x,mean,standard_dev,cumulative)
```

The NORM.DIST function syntax has the following arguments:

- **X** Required. The value for which you want the distribution.
- **Mean** Required. The arithmetic mean of the distribution.
- **Standard_dev** Required. The standard deviation of the distribution.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, NORM.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.

Remarks

- If mean or standard_dev is nonnumeric, NORM.DIST returns the #VALUE! error value.
- If standard_dev ≤ 0, NORM.DIST returns the #NUM! error value.
- If mean = 0, standard_dev = 1, and cumulative = TRUE, NORM.DIST returns the standard normal distribution, NORM.S.DIST.
- The equation for the normal density function (cumulative = FALSE) is:

$$f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\left(\frac{(x-\mu)^2}{2\sigma^2}\right)}$$

- When cumulative = TRUE, the formula is the integral from negative infinity to x of the given formula.

NORM.INV function

Description

Returns the inverse of the normal cumulative distribution for the specified mean and standard deviation.

Syntax

```
NORM.INV(probability,mean,standard_dev)
```

The NORM.INV function syntax has the following arguments:

- **Probability** Required. A probability corresponding to the normal distribution.
- **Mean** Required. The arithmetic mean of the distribution.
- **Standard_dev** Required. The standard deviation of the distribution.

Remarks

- If any argument is nonnumeric, NORM.INV returns the #VALUE! error value.
- If probability ≤ 0 or if probability ≥ 1 , NORM.INV returns the #NUM! error value.
- If standard_dev ≤ 0 , NORM.INV returns the #NUM! error value.
- If mean = 0 and standard_dev = 1, NORM.INV uses the standard normal distribution (see NORMS.INV).

Given a value for probability, NORM.INV seeks that value x such that NORM.DIST(x, mean, standard_dev, TRUE) = probability. Thus, precision of NORM.INV depends on precision of NORM.DIST.

NORM.S.DIST function

Description

Returns the standard normal distribution (has a mean of zero and a standard deviation of one).

Use this function in place of a table of standard normal curve areas.

Syntax

```
NORM.S.DIST(z, cumulative)
```

The NORM.S.DIST function syntax has the following arguments:

- **Z** Required. The value for which you want the distribution.
- **Cumulative** Required. Cumulative is a logical value that determines the form of the function. If cumulative is TRUE, NORMS.DIST returns the cumulative distribution function; if FALSE, it returns the probability mass function.

Remarks

- If z is nonnumeric, NORM.S.DIST returns the #VALUE! error value.
- The equation for the standard normal density function is:

$$f(z) = \frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}}$$

NORM.S.INV function

Description

Returns the inverse of the standard normal cumulative distribution. The distribution has a mean of zero and a standard deviation of one.

Syntax

```
NORM.S.INV(probability)
```

The NORM.S.INV function syntax has the following arguments:

- **Probability** Required. A probability corresponding to the normal distribution.

Remarks

- If probability is nonnumeric, NORMS.INV returns the #VALUE! error value.
- If probability ≤ 0 or if probability ≥ 1 , NORMS.INV returns the #NUM! error value.

Given a value for probability, NORM.S.INV seeks that value z such that $\text{NORM.S.DIST}(z, \text{TRUE}) = \text{probability}$. Thus, precision of NORM.S.INV depends on precision of NORM.S.DIST. NORM.S.INV uses an iterative search technique.

PEARSON function

Description

Returns the Pearson product moment correlation coefficient, r , a dimensionless index that ranges from -1.0 to 1.0 inclusive and reflects the extent of a linear relationship between two data sets.

Syntax

```
PEARSON(array1, array2)
```

The PEARSON function syntax has the following arguments:

- **Array1** Required. A set of independent values.
- **Array2** Required. A set of dependent values.

Remarks

- The arguments must be either numbers or names, array constants, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If array1 and array2 are empty or have a different number of data points, PEARSON returns the #N/A error value.
- The formula for the Pearson product moment correlation coefficient, r , is:

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

where \bar{x} and \bar{y} are the sample means AVERAGE(array1) and AVERAGE(array2).

PERCENTILE.EXC function

Description

Returns the k-th percentile of values in a range, where k is in the range 0..1, exclusive.

Syntax

```
PERCENTILE.EXC(array, k)
```

The PERCENTILE.EXC function syntax has the following arguments:

- **Array** Required. The array or range of data that defines relative standing.
- **K** Required. The percentile value in the range 0..1, exclusive.

Remarks

- If array is empty, PERCENTILE.EXC returns the #NUM! error value
- If k is nonnumeric, PERCENTILE.EXC returns the #VALUE! error value.
- If k is ≤ 0 or if $k \geq 1$, PERCENTILE.EXC returns the #NUM! error value.
- If k is not a multiple of $1/(n - 1)$, PERCENTILE.EXC interpolates to determine the value at the k-th percentile.
- PERCENTILE.EXC will interpolate when the value for the specified percentile lies between two values in the array. If it cannot interpolate for the percentile, k specified, will return #NUM! error.

PERCENTILE.INC function

Description

Returns the k-th percentile of values in a range, where k is in the range 0..1, inclusive.

You can use this function to establish a threshold of acceptance. For example, you can decide to examine candidates who score above the 90th percentile.

Syntax

```
PERCENTILE.INC(array, k)
```

The PERCENTILE.INC function syntax has the following arguments:

- **Array** Required. The array or range of data that defines relative standing.
- **K** Required. The percentile value in the range 0..1, inclusive.

Remarks

- If array is empty, PERCENTILE.INC returns the #NUM! error value.
- If k is nonnumeric, PERCENTILE.INC returns the #VALUE! error value.
- If k is < 0 or if k > 1, PERCENTILE.INC returns the #NUM! error value.
- If k is not a multiple of $1/(n - 1)$, PERCENTILE.INC interpolates to determine the value at the k-th percentile.

PERCENTRANK.EXC function

Description

Returns the rank of a value in a data set as a percentage (0..1, exclusive) of the data set.

Syntax

```
PERCENTRANK.EXC(array, x, [significance])
```

The PERCENTRANK.EXC function syntax has the following arguments:

- **Array** Required. The array or range of data with numeric values that defines relative standing.
- **X** Required. The value for which you want to know the rank.
- **Significance** Optional. A value that identifies the number of significant digits for the returned percentage value. If omitted, PERCENTRANK.EXC uses three digits (0.xxx).

Remarks

- If array is empty, PERCENTRANK.EXC returns the #NUM! error value.
- If significance < 1, PERCENTRANK.EXC returns the #NUM! error value.
- If x does not match one of the values in array, PERCENTRANK.EXC interpolates to return the correct percentage rank.

PERCENTRANK.INC function

Description

Returns the rank of a value in a data set as a percentage (0..1, inclusive) of the data set.

This function can be used to evaluate the relative standing of a value within a data set. For example, you can use PERCENTRANK.INC to evaluate the standing of an aptitude test score among all scores for the test.

Syntax

```
PERCENTRANK.INC(array,x,[significance])
```

The PERCENTRANK.INC function syntax has the following arguments:

- **Array** Required. The array or range of data with numeric values that defines relative standing.
- **X** Required. The value for which you want to know the rank.
- **Significance** Optional. A value that identifies the number of significant digits for the returned percentage value. If omitted, PERCENTRANK.INC uses three digits (0.xxx).

Remarks

- If array is empty, PERCENTRANK.INC returns the #NUM! error value.
- If significance < 1, PERCENTRANK.INC returns the #NUM! error value.
- If x does not match one of the values in array, PERCENTRANK.INC interpolates to return the correct percentage rank.

PERMUT function

Description

Returns the number of permutations for a given number of objects that can be selected from number objects. A permutation is any set or subset of objects or events where internal order is significant. Permutations are different from combinations, for which the internal order is not significant. Use this function for lottery-style probability calculations.

Syntax

```
PERMUT(number, number_chosen)
```

The PERMUT function syntax has the following arguments:

- **Number** Required. An integer that describes the number of objects.
- **Number_chosen** Required. An integer that describes the number of objects in each permutation.

Remarks

- Both arguments are truncated to integers.
- If number or number_chosen is nonnumeric, PERMUT returns the #VALUE! error value.
- If number ≤ 0 or if number_chosen < 0 , PERMUT returns the #NUM! error value.
- If number $<$ number_chosen, PERMUT returns the #NUM! error value.
- The equation for the number of permutations is:

$$P_{k,n} = \frac{n!}{(n-k)!}$$

POISSON.DIST function

Description

Returns the Poisson distribution. A common application of the Poisson distribution is predicting the number of events over a specific time, such as the number of cars arriving at a toll plaza in 1 minute.

Syntax

```
POISSON.DIST(x, mean, cumulative)
```

The POISSON.DIST function syntax has the following arguments:

- **X** Required. The number of events.
- **Mean** Required. The expected numeric value.
- **Cumulative** Required. A logical value that determines the form of the probability distribution returned. If cumulative is TRUE, POISSON.DIST returns the cumulative Poisson probability that the number of random events occurring will be between zero and x inclusive; if FALSE, it returns the Poisson probability mass function that the number of events occurring will be exactly x.

Remarks

- If x is not an integer, it is truncated.
- If x or mean is nonnumeric, POISSON.DIST returns the #VALUE! error value.
- If x < 0, POISSON.DIST returns the #NUM! error value.
- If mean < 0, POISSON.DIST returns the #NUM! error value.
- POISSON.DIST is calculated as follows.

For cumulative = FALSE:

$$POISSON = \frac{e^{-\lambda} \lambda^x}{x!}$$

For cumulative = TRUE:

$$CUMPOISSON = \sum_{k=0}^x \frac{e^{-\lambda} \lambda^k}{k!}$$

PROB function

Description

Returns the probability that values in a range are between two limits. If upper_limit is not supplied, returns the probability that values in x_range are equal to lower_limit.

Syntax

```
PROB(x_range, prob_range, [lower_limit], [upper_limit])
```

The PROB function syntax has the following arguments:

- **X_range** Required. The range of numeric values of x with which there are associated probabilities.
- **Prob_range** Required. A set of probabilities associated with values in x_range.
- **Lower_limit** Optional. The lower bound on the value for which you want a probability.
- **Upper_limit** Optional. The optional upper bound on the value for which you want a probability.

Remarks

- If any value in prob_range ≤ 0 or if any value in prob_range > 1 , PROB returns the #NUM! error value.
- If the sum of the values in prob_range is not equal to 1, PROB returns the #NUM! error value.
- If upper_limit is omitted, PROB returns the probability of being equal to lower_limit.
- If x_range and prob_range contain a different number of data points, PROB returns the #N/A error value.

QUARTILE.EXC function

Description

Returns the quartile of the data set, based on percentile values from 0..1, exclusive.

Syntax

```
QUARTILE.EXC(array, quart)
```

The QUARTILE.EXC function syntax has the following arguments:

- **Array** Required. The array or cell range of numeric values for which you want the quartile value.
- **Quart** Required. Indicates which value to return.

Remarks

- If array is empty, QUARTILE.EXC returns the #NUM! error value.
- If quart is not an integer, it is truncated.
- If $\text{quart} \leq 0$ or if $\text{quart} \geq 4$, QUARTILE.EXC returns the #NUM! error value.
- MIN, MEDIAN, and MAX return the same value as QUARTILE.EXC when quart is equal to 0 (zero), 2, and 4, respectively.

QUARTILE.INC function

Description

Returns the quartile of a data set, based on percentile values from 0..1, inclusive.

Quartiles often are used in sales and survey data to divide populations into groups. For example, you can use QUARTILE.INC to find the top 25 percent of incomes in a population.

Syntax

```
QUARTILE.INC(array, quart)
```

The QUARTILE.INC function syntax has the following arguments:

- **Array** Required. The array or cell range of numeric values for which you want the quartile value.
- **Quart** Required. Indicates which value to return.

Parameters:

If quart equals	QUARTILE.INC returns
0	Minimum value
1	First quartile (25th percentile)
2	Median value (50th percentile)
3	Third quartile (75th percentile)
4	Maximum value

Remarks

- If array is empty, QUARTILE.INC returns the #NUM! error value.
- If quart is not an integer, it is truncated.
- If quart < 0 or if quart > 4, QUARTILE.INC returns the #NUM! error value.

- MIN, MEDIAN, and MAX return the same value as QUARTILE.INC when quart is equal to 0 (zero), 2, and 4, respectively.

RANK.AVG function

Description

Returns the rank of a number in a list of numbers: its size relative to other values in the list; if more than one value has the same rank, the average rank is returned.

Syntax

```
RANK.AVG(number, ref, [order])
```

The RANK.AVG function syntax has the following arguments:

- **Number** Required. The number whose rank you want to find.
- **Ref** Required. An array of, or a reference to, a list of numbers. Nonnumeric values in ref are ignored.
- **Order** Optional. A number specifying how to rank number.

If order is 0 (zero) or omitted, ranks number as if ref were a list sorted in descending order.

If order is any nonzero value, ranks number as if ref were a list sorted in ascending order.

RANK.EQ function

Description

Returns the rank of a number in a list of numbers. Its size is relative to other values in the list; if more than one value has the same rank, the top rank of that set of values is returned.

If you were to sort the list, the rank of the number would be its position.

Syntax

```
RANK.EQ(number, ref, [order])
```

The RANK.EQ function syntax has the following arguments:

- **Number** Required. The number whose rank you want to find.
- **Ref** Required. An array of, or a reference to, a list of numbers. Nonnumeric values in ref are ignored.
- **Order** Optional. A number specifying how to rank number.

If order is 0 (zero) or omitted, ranks number as if ref were a list sorted in descending order.

If order is any nonzero value, ranks number as if ref were a list sorted in ascending order.

Remarks

- RANK.EQ gives duplicate numbers the same rank. However, the presence of duplicate numbers affects the ranks of subsequent numbers. For example, in a list of integers sorted in ascending order, if the number 10 appears twice and has a rank of 5, then 11 would have a rank of 7 (no number would have a rank of 6).
- For some purposes one might want to use a definition of rank that takes ties into account. In the previous example, one would want a revised rank of 5.5 for the number 10. This can be done by adding the following correction factor to the value returned by RANK.EQ. This correction factor is

appropriate both for the case where rank is computed in descending order (order = 0 or omitted) or ascending order (order = nonzero value).

- Correction factor for tied ranks= $[\text{COUNT}(\text{ref}) + 1 - \text{RANK.EQ}(\text{number}, \text{ref}, 0) - \text{RANK.EQ}(\text{number}, \text{ref}, 1)]/2$.

In the following example, $\text{RANK.EQ}(A2,A1:A5,1)$ equals 3. The correction factor is $(5 + 1 - 2 - 3)/2 = 0.5$ and the revised rank that takes ties into account is $3 + 0.5 = 3.5$. If number occurs only once in ref, the correction factor will be 0, since RANK.EQ would not have to be adjusted for a tie.

RSQ function

Description

Returns the square of the Pearson product moment correlation coefficient through data points in known_y's and known_x's. For more information, see the [PEARSON function](#). The r-squared value can be interpreted as the proportion of the variance in y attributable to the variance in x.

Syntax

```
RSQ (known_y's, known_x's)
```

The RSQ function syntax has the following arguments:

- **Known_y's** Required. An array or range of data points.
- **Known_x's** Required. An array or range of data points.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If known_y's and known_x's are empty or have a different number of data points, RSQ returns the #N/A error value.
- If known_y's and known_x's contain only 1 data point, RSQ returns the #DIV/0! error value.
- The equation for the Pearson product moment correlation coefficient, r, is:

$$r = \frac{\sum (x - \bar{x})(y - \bar{y})}{\sqrt{\sum (x - \bar{x})^2 \sum (y - \bar{y})^2}}$$

where x and y are the sample means $AVERAGE(\text{known_x's})$ and $AVERAGE(\text{known_y's})$.

RSQ returns r^2 , which is the square of this correlation coefficient.

SKEW function

Description

Returns the skewness of a distribution. Skewness characterizes the degree of asymmetry of a distribution around its mean. Positive skewness indicates a distribution with an asymmetric tail extending toward more positive values. Negative skewness indicates a distribution with an asymmetric tail extending toward more negative values.

Syntax

```
SKEW(number1, [number2], ...)
```

The SKEW function syntax has the following arguments:

- **Number1, number2, ...** Number1 is required, subsequent numbers are optional. 1 to 255 arguments for which you want to calculate skewness. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If there are fewer than three data points, or the sample standard deviation is zero, SKEW returns the #DIV/0! error value.
- The equation for skewness is defined as:

$$\frac{n}{(n-1)(n-2)} \sum \left(\frac{x_j - \bar{x}}{s} \right)^3$$

SLOPE function

Description

Returns the slope of the linear regression line through data points in known_y's and known_x's. The slope is the vertical distance divided by the horizontal distance between any two points on the line, which is the rate of change along the regression line.

Syntax

```
SLOPE(known_y's, known_x's)
```

The SLOPE function syntax has the following arguments:

- **Known_y's** Required. An array or cell range of numeric dependent data points.
- **Known_x's** Required. The set of independent data points.

Remarks

- The arguments must be either numbers or names, arrays, or references that contain numbers.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- If known_y's and known_x's are empty or have a different number of data points, SLOPE returns the #N/A error value.
- The equation for the slope of the regression line is:

$$b = \frac{\sum(x - \bar{x})(y - \bar{y})}{\sum(x - \bar{x})^2}$$

where x and y are the sample means AVERAGE(known_x's) and AVERAGE(known_y's).

- The underlying algorithm used in the SLOPE and INTERCEPT functions is different than the underlying algorithm used in the LINEST function. The difference between these algorithms can

lead to different results when data is undetermined and collinear. For example, if the data points of the known_y's argument are 0 and the data points of the known_x's argument are 1:

- SLOPE and INTERCEPT return a #DIV/0! error. The SLOPE and INTERCEPT algorithm is designed to look for one and only one answer, and in this case there can be more than one answer.
- LINEST returns a value of 0. The LINEST algorithm is designed to return reasonable results for collinear data, and in this case at least one answer can be found.

SMALL function

Description

Returns the k-th smallest value in a data set. Use this function to return values with a particular relative standing in a data set.

Syntax

```
SMALL(array, k)
```

The SMALL function syntax has the following arguments:

- **Array** Required. An array or range of numerical data for which you want to determine the k-th smallest value.
- **K** Required. The position (from the smallest) in the array or range of data to return.

Remarks

- If array is empty, SMALL returns the #NUM! error value.
- If $k \leq 0$ or if k exceeds the number of data points, SMALL returns the #NUM! error value.
- If n is the number of data points in array, SMALL(array,1) equals the smallest value, and SMALL(array,n) equals the largest value.

STANDARDIZE function

Description

Returns a normalized value from a distribution characterized by mean and standard_dev.

Syntax

```
STANDARDIZE(x, mean, standard_dev)
```

The STANDARDIZE function syntax has the following arguments:

- **X** Required. The value you want to normalize.
- **Mean** Required. The arithmetic mean of the distribution.
- **Standard_dev** Required. The standard deviation of the distribution.

Remarks

- If standard_dev ≤ 0, STANDARDIZE returns the #NUM! error value.
- The equation for the normalized value is:

$$Z = \frac{X - \mu}{\sigma}$$

STDEV.P function

Description

Calculates standard deviation based on the entire population given as arguments (ignores logical values and text).

The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

Syntax

```
STDEV.P(number1, [number2], ...)
```

The STDEV.P function syntax has the following arguments:

- **Number1** Required. The first number argument corresponding to a population.
- **Number2, ...** Optional. Number arguments 2 to 254 corresponding to a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- STDEV.P assumes that its arguments are the entire population. If your data represents a sample of the population, then compute the standard deviation using STDEV.
- For large sample sizes, STDEV.S and STDEV.P return approximately equal values.
- The standard deviation is calculated using the "n" method.
- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values, and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.

- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the STDEVPA function.
- STDEV.P uses the following formula:

$$\sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

where \bar{x} is the sample mean AVERAGE(number1,number2,...) and n is the sample size.

STDEV.S function

Description

Estimates standard deviation based on a sample (ignores logical values and text in the sample).

The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

Syntax

```
STDEV.S(number1, [number2], ...)
```

The STDEV.S function syntax has the following arguments:

- **Number1** Required. The first number argument corresponding to a sample of a population. You can also use a single array or a reference to an array instead of arguments separated by commas.
- **Number2, ...** Optional. Number arguments 2 to 254 corresponding to a sample of a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- STDEV.S assumes that its arguments are a sample of the population. If your data represents the entire population, then compute the standard deviation using STDEV.P.
- The standard deviation is calculated using the "n-1" method.
- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.

- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the STDEVA function.
- STDEV.S uses the following formula:

$$\sqrt{\frac{\sum (x - \bar{x})^2}{(n - 1)}}$$

where \bar{x} is the sample mean AVERAGE(number1,number2,...) and n is the sample size.

STDEVA function

Description

Estimates standard deviation based on a sample. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

Syntax

```
STDEVA(value1, [value2], ...)
```

The STDEVA function syntax has the following arguments:

- **Value1, value2, ...** Value1 is required, subsequent values are optional. 1 to 255 values corresponding to a sample of a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- STDEVA assumes that its arguments are a sample of the population. If your data represents the entire population, you must compute the standard deviation using STDEVPA.
- The standard deviation is calculated using the "n-1" method.
- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the STDEV function.
- STDEVA uses the following formula:

$$\sqrt{\frac{\sum (x - \bar{x})^2}{(n-1)}}$$

where \bar{x} is the sample mean `AVERAGE(value1,value2,...)` and n is the sample size.

STDEVPA function

Description

Calculates standard deviation based on the entire population given as arguments, including text and logical values. The standard deviation is a measure of how widely values are dispersed from the average value (the mean).

Syntax

```
STDEVPA(value1, [value2], ...)
```

The STDEVPA function syntax has the following arguments:

- **Value1, value2, ...** Value1 is required, subsequent values are optional. 1 to 255 values corresponding to a population. You can also use a single array or a reference to an array instead of arguments separated by commas.

Remarks

- STDEVPA assumes that its arguments are the entire population. If your data represents a sample of the population, you must compute the standard deviation by using STDEVA.
- For large sample sizes, STDEVA and STDEVPA return approximately equal values.
- The standard deviation is calculated using the "n" method.
- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- Text representations of numbers that you type directly into the list of arguments are counted.
- Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.

- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the STDEVP function.
- STDEVPA uses the following formula:

$$\sqrt{\frac{\sum (x - \bar{x})^2}{n}}$$

where \bar{x} is the sample mean `AVERAGE(value1,value2,...)` and n is the sample size.

STEYX function

Description

Returns the standard error of the predicted y-value for each x in the regression. The standard error is a measure of the amount of error in the prediction of y for an individual x.

Syntax

```
STEYX(known_y's, known_x's)
```

The STEYX function syntax has the following arguments:

- **Known_y's** Required. An array or range of dependent data points.
- **Known_x's** Required. An array or range of independent data points.

Remarks

- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- If an array or reference argument contains text, logical values, or empty cells, those values are ignored; however, cells with the value zero are included.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If known_y's and known_x's have a different number of data points, STEYX returns the #N/A error value.
- If known_y's and known_x's are empty or have less than three data points, STEYX returns the #DIV/0! error value.
- The equation for the standard error of the predicted y is:

$$\sqrt{\frac{1}{(n-2)} \left[\sum (y-\bar{y})^2 - \frac{[\sum (x-\bar{x})(y-\bar{y})]^2}{\sum (x-\bar{x})^2} \right]}$$

where x and y are the sample means $AVERAGE(\text{known_x's})$ and $AVERAGE(\text{known_y's})$, and n is the sample size.

T.DIST function

Description

Returns the Student's t-distribution. The t-distribution is used in the hypothesis testing of small sample data sets. Use this function in place of a table of critical values for the t-distribution.

Syntax

```
T.DIST(x,deg_freedom, cumulative)
```

The T.DIST function syntax has the following arguments:

- **X** Required. The numeric value at which to evaluate the distribution
- **Deg_freedom** Required. An integer indicating the number of degrees of freedom.
- **Cumulative** Required. A logical value that determines the form of the function. If cumulative is TRUE, T.DIST returns the cumulative distribution function; if FALSE, it returns the probability density function.

Remarks

- If any argument is nonnumeric, T.DIST returns the #VALUE! error value.
- If deg_freedom < 1, T.DIST returns the #NUM! error value.
- If x < 0, then T.DIST returns the #NUM! error value.

T.DIST.2T function

Description

Returns the two-tailed Student's t-distribution.

The Student's t-distribution is used in the hypothesis testing of small sample data sets. Use this function in place of a table of critical values for the t-distribution.

Syntax

```
T.DIST.2T(x, deg_freedom)
```

The T.DIST.2T function syntax has the following arguments:

- **X** Required. The numeric value at which to evaluate the distribution.
- **Deg_freedom** Required. An integer indicating the number of degrees of freedom.

Remarks

- If any argument is nonnumeric, T.DIST.2T returns the #VALUE! error value.
- If $\text{deg_freedom} < 1$, T.DIST.2T returns the #NUM! error value.
- If $x < 0$, then T.DIST.2T returns the #NUM! error value.

T.DIST.RT function

Description

Returns the right-tailed Student's t-distribution.

The t-distribution is used in the hypothesis testing of small sample data sets. Use this function in place of a table of critical values for the t-distribution.

Syntax

```
T.DIST.RT(x, deg_freedom)
```

The T.DIST.RT function syntax has the following arguments:

- **X** Required. The numeric value at which to evaluate the distribution.
- **Deg_freedom** Required. An integer indicating the number of degrees of freedom.

Remarks

- If any argument is nonnumeric, T.DIST.RT returns the #VALUE! error value.
- If $\text{deg_freedom} < 1$, T.DIST.RT returns the #NUM! error value.
- If $x < 0$, then T.DIST.RT returns the #NUM! error value.

T.INV function

Description

Returns the left-tailed inverse of the Student's t-distribution.

Syntax

```
T.INV(probability,deg_freedom)
```

The T.INV function syntax has the following arguments:

- **Probability** Required. The probability associated with the Student's t-distribution.
- **Deg_freedom** Required. The number of degrees of freedom with which to characterize the distribution.

Remarks

- If either argument is nonnumeric, T.INV returns the #VALUE! error value.
- If probability ≤ 0 or if probability > 1 , T.INV returns the #NUM! error value.
- If deg_freedom is not an integer, it is truncated.
- If deg_freedom < 1 , T.INV returns the #NUM! error value.

T.INV.2T function

Description

Returns the two-tailed inverse of the Student's t-distribution.

Syntax

```
T.INV.2T(probability,deg_freedom)
```

The T.INV.2T function syntax has the following arguments:

- **Probability** Required. The probability associated with the Student's t-distribution.
- **Deg_freedom** Required. The number of degrees of freedom with which to characterize the distribution.

Remarks

- If either argument is nonnumeric, T.INV.2T returns the #VALUE! error value.
- If probability ≤ 0 or if probability > 1 , T.INV.2T returns the #NUM! error value.
- If deg_freedom is not an integer, it is truncated.
- If deg_freedom < 1 , T.INV.2T returns the #NUM! error value.
- T.INV.2T returns that value t , such that $P(|X| > t) = \text{probability}$ where X is a random variable that follows the t-distribution and $P(|X| > t) = P(X < -t \text{ or } X > t)$.
- A one-tailed t-value can be returned by replacing probability with $2 * \text{probability}$. For a probability of 0.05 and degrees of freedom of 10, the two-tailed value is calculated with T.INV.2T(0.05,10), which returns 2.28139. The one-tailed value for the same probability and degrees of freedom can be calculated with T.INV.2T($2 * 0.05$,10), which returns 1.812462.
- Given a value for probability, T.INV.2T seeks that value x such that T.DIST.2T(x , deg_freedom, 2) = probability. Thus, precision of T.INV.2T depends on precision of T.DIST.2T.

TREND function

Description

Returns values along a linear trend. Fits a straight line (using the method of least squares) to the arrays `known_y's` and `known_x's`. Returns the y-values along that line for the array of `new_x's` that you specify.

Syntax

```
TREND(known_y's, [known_x's], [new_x's], [const])
```

The TREND function syntax has the following arguments:

- **Known_y's** Required. The set of y-values you already know in the relationship $y = mx + b$.
 - If the array `known_y's` is in a single column, then each column of `known_x's` is interpreted as a separate variable.
 - If the array `known_y's` is in a single row, then each row of `known_x's` is interpreted as a separate variable.
- **Known_x's** Required. An optional set of x-values that you may already know in the relationship $y = mx + b$.
 - The array `known_x's` can include one or more sets of variables. If only one variable is used, `known_y's` and `known_x's` can be ranges of any shape, as long as they have equal dimensions. If more than one variable is used, `known_y's` must be a vector (that is, a range with a height of one row or a width of one column).
 - If `known_x's` is omitted, it is assumed to be the array {1,2,3,...} that is the same size as `known_y's`.
- **New_x's** Required. New x-values for which you want TREND to return corresponding y-values.
 - `New_x's` must include a column (or row) for each independent variable, just as `known_x's` does. So, if `known_y's` is in a single column, `known_x's` and `new_x's` must have the same

number of columns. If known_y's is in a single row, known_x's and new_x's must have the same number of rows.

- If you omit new_x's, it is assumed to be the same as known_x's.
- If you omit both known_x's and new_x's, they are assumed to be the array {1,2,3,...} that is the same size as known_y's.
- **Const** Optional. A logical value specifying whether to force the constant b to equal 0.
 - If const is TRUE or omitted, b is calculated normally.
 - If const is FALSE, b is set equal to 0 (zero), and the m-values are adjusted so that $y = mx$.

Remarks

- You can use TREND for polynomial curve fitting by regressing against the same variable raised to different powers. For example, suppose column A contains y-values and column B contains x-values. You can enter x^2 in column C, x^3 in column D, and so on, and then regress columns B through D against column A.
- Formulas that return arrays must be entered as array formulas.
- When entering an array constant for an argument such as known_x's, use commas to separate values in the same row and semicolons to separate rows.

NOTE The formula in the example must be entered as an array formula. After copying the example to a blank worksheet, select the range C2:C13 or B15:B19 starting with the formula cell. Press F2, and then press CTRL+SHIFT+ENTER. If the formula is not entered as an array formula, the single results are 133953.3333 and 146171.5152.

TRIMMEAN function

Description

Returns the mean of the interior of a data set. TRIMMEAN calculates the mean taken by excluding a percentage of data points from the top and bottom tails of a data set. You can use this function when you wish to exclude outlying data from your analysis.

Syntax

```
TRIMMEAN(array, percent)
```

The TRIMMEAN function syntax has the following arguments:

- **Array** Required. The array or range of values to trim and average.
- **Percent** Required. The fractional number of data points to exclude from the calculation. For example, if percent = 0.2, 4 points are trimmed from a data set of 20 points (20×0.2): 2 from the top and 2 from the bottom of the set.

Remarks

- If percent < 0 or percent > 1, TRIMMEAN returns the #NUM! error value.
- TRIMMEAN rounds the number of excluded data points down to the nearest multiple of 2. If percent = 0.1, 10 percent of 30 data points equals 3 points. For symmetry, TRIMMEAN excludes a single value from the top and bottom of the data set.

T.TEST function

Description

Returns the probability associated with a Student's t-Test. Use T.TEST to determine whether two samples are likely to have come from the same two underlying populations that have the same mean.

Syntax

```
T.TEST(array1, array2, tails, type)
```

The T.TEST function syntax has the following arguments:

- **Array1** Required. The first data set.
- **Array2** Required. The second data set.
- **Tails** Required. Specifies the number of distribution tails. If tails = 1, T.TEST uses the one-tailed distribution. If tails = 2, T.TEST uses the two-tailed distribution.
- **Type** Required. The kind of t-Test to perform.

Parameters:

If type equals	This test is performed
1	Paired
2	Two-sample equal variance (homoscedastic)
3	Two-sample unequal variance (heteroscedastic)

Remarks

- If array1 and array2 have a different number of data points, and type = 1 (paired), T.TEST returns the #N/A error value.
- The tails and type arguments are truncated to integers.
- If tails or type is nonnumeric, T.TEST returns the #VALUE! error value.
- If tails is any value other than 1 or 2, T.TEST returns the #NUM! error value.

- T.TEST uses the data in array1 and array2 to compute a non-negative t-statistic. If tails=1, T.TEST returns the probability of a higher value of the t-statistic under the assumption that array1 and array2 are samples from populations with the same mean. The value returned by T.TEST when tails=2 is double that returned when tails=1 and corresponds to the probability of a higher absolute value of the t-statistic under the “same population means” assumption.

VAR.P function

Description

Calculates variance based on the entire population (ignores logical values and text in the population).

Syntax

```
VAR.P(number1, [number2], ...)
```

The VAR.P function syntax has the following arguments:

- **Number1** Required. The first number argument corresponding to a population.
- **Number2, ...** Optional. Number arguments 2 to 254 corresponding to a population.

Remarks

- VAR.P assumes that its arguments are the entire population. If your data represents a sample of the population, then compute the variance by using VAR.S.
- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values, and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the VARPA function.
- The equation for VAR.P is:

$$\frac{\sum (x - \bar{x})^2}{n}$$

where \bar{x} is the sample mean AVERAGE(number1,number2,...) and n is the sample size.

VAR.S function

Description

Estimates variance based on a sample (ignores logical values and text in the sample).

Syntax

```
VAR.S(number1, [number2], ...)
```

The VAR.S function syntax has the following arguments:

- **Number1** Required. The first number argument corresponding to a sample of a population.
- **Number2, ...** Optional. Number arguments 2 to 254 corresponding to a sample of a population.

Remarks

- VAR.S assumes that its arguments are a sample of the population. If your data represents the entire population, then compute the variance by using VAR.P.
- Arguments can either be numbers or names, arrays, or references that contain numbers.
- Logical values, and text representations of numbers that you type directly into the list of arguments are counted.
- If an argument is an array or reference, only numbers in that array or reference are counted. Empty cells, logical values, text, or error values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you want to include logical values and text representations of numbers in a reference as part of the calculation, use the VARA function.
- VAR.S uses the following formula:

$$\frac{\sum (x - \bar{x})^2}{(n - 1)}$$

where \bar{x} is the sample mean AVERAGE(number1,number2,...) and n is the sample size.

VARA function

Description

Estimates variance based on a sample.

Syntax

```
VARA(value1, [value2], ...)
```

The VARA function syntax has the following arguments:

- **Value1, value2, ...** Value1 is required, subsequent values are optional. 1 to 255 value arguments corresponding to a sample of a population.

Remarks

- VARA assumes that its arguments are a sample of the population. If your data represents the entire population, you must compute the variance by using VARPA.
- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the VAR function.
- VARA uses the following formula:

$$\frac{\sum (x - \bar{x})^2}{(n - 1)}$$

where \bar{x} is the sample mean `AVERAGE(value1,value2,...)` and n is the sample size.

VARPA function

Description

Calculates variance based on the entire population.

Syntax

```
VARPA(value1, [value2], ...)
```

The VARPA function syntax has the following arguments:

- **Value1, value2, ...** Value1 is required, subsequent values are optional. 1 to 255 value arguments corresponding to a population.

Remarks

- VARPA assumes that its arguments are the entire population. If your data represents a sample of the population, you must compute the variance by using VARA.
- Arguments can be the following: numbers; names, arrays, or references that contain numbers; text representations of numbers; or logical values, such as TRUE and FALSE, in a reference.
- Logical values and text representations of numbers that you type directly into the list of arguments are counted.
- Arguments that contain TRUE evaluate as 1; arguments that contain text or FALSE evaluate as 0 (zero).
- If an argument is an array or reference, only values in that array or reference are used. Empty cells and text values in the array or reference are ignored.
- Arguments that are error values or text that cannot be translated into numbers cause errors.
- If you do not want to include logical values and text representations of numbers in a reference as part of the calculation, use the VARP function.
- The equation for VARPA is :

$$\frac{\sum(x-\bar{x})^2}{n}$$

where \bar{x} is the sample mean `AVERAGE(value1,value2,...)` and n is the sample size.

WEIBULL.DIST function

Description

Returns the Weibull distribution. Use this distribution in reliability analysis, such as calculating a device's mean time to failure.

Syntax

```
WEIBULL.DIST(x, alpha, beta, cumulative)
```

The WEIBULL.DIST function syntax has the following arguments:

- **X** Required. The value at which to evaluate the function.
- **Alpha** Required. A parameter to the distribution.
- **Beta** Required. A parameter to the distribution.
- **Cumulative** Required. Determines the form of the function.

Remarks

- If x, alpha, or beta is nonnumeric, WEIBULL.DIST returns the #VALUE! error value.
- If $x < 0$, WEIBULL.DIST returns the #NUM! error value.
- If $\alpha \leq 0$ or if $\beta \leq 0$, WEIBULL.DIST returns the #NUM! error value.
- The equation for the Weibull cumulative distribution function is:

$$F(x; \alpha, \beta) = 1 - e^{-(x/\beta)^\alpha}$$

- The equation for the Weibull probability density function is:

$$f(x; \alpha, \beta) = \frac{\alpha}{\beta^\alpha} x^{\alpha-1} e^{-(x/\beta)^\alpha}$$

- When $\alpha = 1$, WEIBULL.DIST returns the exponential distribution with:

$$\lambda = \frac{1}{\beta}$$

Z.TEST function

Description

Returns the one-tailed P-value of a z-test.

For a given hypothesized population mean, *x*, Z.TEST returns the probability that the sample mean would be greater than the average of observations in the data set (*array*) — that is, the observed sample mean.

To see how Z.TEST can be used in a formula to compute a two-tailed probability value, see the Remarks section below.

Syntax

```
Z.TEST(array,x,[sigma])
```

The Z.TEST function syntax has the following arguments:

- **Array** Required. The array or range of data against which to test *x*.
- **x** Required. The value to test.
- **Sigma** Optional. The population (known) standard deviation. If omitted, the sample standard deviation is used.

Remarks

- If *array* is empty, Z.TEST returns the #N/A error value.
- Z.TEST is calculated as follows when *sigma* is not omitted:

$$Z.TEST(array,x,sigma) = 1 - \text{Norm.S.Dist}((\text{Average}(array) - x) / (sigma/\sqrt{n}), TRUE)$$

or when *sigma* is omitted:

$$Z.TEST(array,x) = 1 - \text{Norm.S.Dist}((\text{Average}(array) - x) / (\text{STDEV}(array)/\sqrt{n}), TRUE)$$

where *x* is the sample mean AVERAGE(*array*), and *n* is COUNT(*array*).

- Z.TEST represents the probability that the sample mean would be greater than the observed value AVERAGE(array), when the underlying population mean is μ_0 . From the symmetry of the Normal distribution, if AVERAGE(array) < x, Z.TEST will return a value greater than 0.5.
- The following formula can be used to calculate the two-tailed probability that the sample mean would be further from x (in either direction) than AVERAGE(array), when the underlying population mean is x:

$$=2 * \text{MIN}(\text{Z.TEST}(\text{array},x,\text{sigma}), 1 - \text{Z.TEST}(\text{array},x,\text{sigma})).$$