

Projet : Valorisation et couverture d'options

Généré par Doxygen 1.14.0

1 Index des espaces de nommage	1
1.1 Liste des espaces de nommage	1
2 Index hiérarchique	3
2.1 Hiérarchie des classes	3
3 Index des classes	5
3.1 Liste des classes	5
4 Index des fichiers	7
4.1 Liste des fichiers	7
5 Documentation des espaces de nommage	9
5.1 Référence de l'espace de nommage crr	9
5.2 Référence de l'espace de nommage opt	10
5.3 Référence de l'espace de nommage pde	10
6 Documentation des classes	13
6.1 Référence de la classe crr::Aggregator	13
6.1.1 Description détaillée	14
6.1.2 Documentation des fonctions membres	14
6.1.2.1 operator()()	14
6.2 Référence du modèle de la classe crr::American< TPayoff >	14
6.2.1 Description détaillée	18
6.2.2 Documentation des constructeurs et destructeur	18
6.2.2.1 American()	18
6.2.3 Documentation des fonctions membres	19
6.2.3.1 deltaRR()	19
6.2.3.2 hedgingStrategy()	19
6.2.3.3 priceRR()	20
6.2.3.4 treePrice()	20
6.3 Référence de la classe crr::Arithmetic	21
6.3.1 Description détaillée	22
6.3.2 Documentation des fonctions membres	22
6.3.2.1 operator()()	22
6.4 Référence du modèle de la classe crr::Asian< TPayoff, TAggregator >	23
6.4.1 Description détaillée	27
6.4.2 Documentation des constructeurs et destructeur	27
6.4.2.1 Asian()	27
6.4.3 Documentation des fonctions membres	28
6.4.3.1 deltaMC()	28
6.4.3.2 hedgingStrategy()	28
6.4.3.3 priceMC()	29
6.4.3.4 terminalValues()	29

6.4.3.5 treePrice()	30
6.5 Référence de la classe pde::BSVol	30
6.5.1 Description détaillée	32
6.5.2 Documentation des constructeurs et destructeur	32
6.5.2.1 BSVol()	32
6.5.3 Documentation des fonctions membres	32
6.5.3.1 operator()()	32
6.6 Référence du modèle de la classe pde::CNMethod< TPDE >	33
6.6.1 Description détaillée	36
6.6.2 Documentation des constructeurs et destructeur	36
6.6.2.1 CNMethod()	36
6.6.3 Documentation des fonctions membres	37
6.6.3.1 A()	37
6.6.3.2 B()	38
6.6.3.3 C()	38
6.6.3.4 D()	39
6.6.3.5 E()	40
6.6.3.6 F()	40
6.6.3.7 G()	40
6.7 Référence du modèle de la classe pde::Diffusion< TPayoff, TVol >	41
6.7.1 Description détaillée	43
6.7.2 Documentation des constructeurs et destructeur	44
6.7.2.1 Diffusion()	44
6.7.3 Documentation des fonctions membres	44
6.7.3.1 a()	44
6.7.3.2 b()	44
6.7.3.3 c()	44
6.7.3.4 d()	44
6.7.3.5 Lower()	45
6.7.3.6 Terminal()	45
6.7.3.7 Upper()	45
6.8 Référence du modèle de la classe crr::European< TPayoff >	46
6.8.1 Description détaillée	48
6.8.2 Documentation des constructeurs et destructeur	49
6.8.2.1 European()	49
6.8.3 Documentation des fonctions membres	49
6.8.3.1 hedgingStrategy()	49
6.8.3.2 treePrice()	50
6.9 Référence du modèle de la classe pde::FDMMethod< TPDE >	51
6.9.1 Description détaillée	54
6.9.2 Documentation des constructeurs et destructeur	55
6.9.2.1 FDMMethod()	55

6.9.3 Documentation des fonctions membres	55
6.9.3.1 a()	55
6.9.3.2 b()	56
6.9.3.3 c()	56
6.9.3.4 d()	56
6.9.3.5 delta()	57
6.9.3.6 f()	57
6.9.3.7 fl()	58
6.9.3.8 fu()	58
6.9.3.9 grid()	58
6.9.3.10 S()	59
6.9.3.11 t()	59
6.9.3.12 v()	60
6.9.4 Documentation des données membres	60
6.9.4.1 dS_	60
6.9.4.2 dt_	60
6.9.4.3 imax_	61
6.9.4.4 jmax_	61
6.9.4.5 pde_	61
6.9.4.6 V	61
6.10 Référence de la classe crr::Geometric	61
6.10.1 Description détaillée	62
6.10.2 Documentation des fonctions membres	62
6.10.2.1 operator()()	62
6.11 Référence de la structure pde::FDMMethod< TPDE >::Grid	63
6.11.1 Description détaillée	63
6.11.2 Documentation des données membres	64
6.11.2.1 del	64
6.11.2.2 val	64
6.12 Référence de la structure crr::Option::HedgingStrategy	64
6.12.1 Description détaillée	65
6.12.2 Documentation des données membres	65
6.12.2.1 bond	65
6.12.2.2 delta	65
6.13 Référence du modèle de la classe pde::ImplicitScheme< TPDE >	65
6.13.1 Description détaillée	69
6.13.2 Documentation des constructeurs et destructeur	69
6.13.2.1 ImplicitScheme()	69
6.13.3 Documentation des fonctions membres	70
6.13.3.1 A() [1/2]	70
6.13.3.2 A() [2/2]	70
6.13.3.3 B()	71

6.13.3.4 C()	71
6.13.3.5 D()	72
6.13.3.6 E()	72
6.13.3.7 F()	73
6.13.3.8 G()	73
6.13.3.9 LUDecomposition()	73
6.13.3.10 SolvePDE()	74
6.13.3.11 w()	74
6.14 Référence de la classe pde::LocalVol	75
6.14.1 Description détaillée	77
6.14.2 Documentation des constructeurs et destructeur	77
6.14.2.1 LocalVol()	77
6.14.3 Documentation des fonctions membres	77
6.14.3.1 operator()()	77
6.15 Référence de la classe crr::LookMax	78
6.15.1 Description détaillée	79
6.15.2 Documentation des fonctions membres	79
6.15.2.1 operator()()	79
6.16 Référence de la classe crr::LookMin	79
6.16.1 Description détaillée	81
6.16.2 Documentation des fonctions membres	81
6.16.2.1 operator()()	81
6.17 Référence de la classe crr::Option	81
6.17.1 Description détaillée	84
6.17.2 Documentation des constructeurs et destructeur	84
6.17.2.1 Option()	84
6.17.3 Documentation des fonctions membres	85
6.17.3.1 deltaZero()	85
6.17.3.2 hedgingStrategy()	85
6.17.3.3 price()	86
6.17.3.4 treePrice()	86
6.17.4 Documentation des données membres	87
6.17.4.1 d_	87
6.17.4.2 discount_	87
6.17.4.3 N_	87
6.17.4.4 R_	87
6.17.4.5 S0_	87
6.17.4.6 sigma_	87
6.17.4.7 T_	88
6.17.4.8 u_	88
6.18 Référence de la classe pde::ParabPDE	88
6.18.1 Description détaillée	90

6.18.2 Documentation des constructeurs et destructeur	90
6.18.2.1 ParabPDE()	90
6.18.3 Documentation des fonctions membres	91
6.18.3.1 a()	91
6.18.3.2 b()	91
6.18.3.3 c()	91
6.18.3.4 d()	92
6.18.3.5 Lower()	92
6.18.3.6 Smax()	92
6.18.3.7 Smin()	93
6.18.3.8 T()	93
6.18.3.9 Terminal()	93
6.18.3.10 Upper()	94
6.18.4 Documentation des données membres	94
6.18.4.1 Smax_	94
6.18.4.2 Smin_	94
6.18.4.3 T_	94
6.19 Référence de la classe opt::Payoff	95
6.19.1 Description détaillée	95
6.19.2 Documentation des fonctions membres	95
6.19.2.1 operator()()	95
6.20 Référence de la classe opt::PayoffBear	96
6.20.1 Description détaillée	97
6.20.2 Documentation des constructeurs et destructeur	97
6.20.2.1 PayoffBear()	97
6.20.3 Documentation des fonctions membres	97
6.20.3.1 operator()()	97
6.21 Référence de la classe opt::PayoffBull	98
6.21.1 Description détaillée	99
6.21.2 Documentation des constructeurs et destructeur	99
6.21.2.1 PayoffBull()	99
6.21.3 Documentation des fonctions membres	99
6.21.3.1 operator()()	99
6.22 Référence de la classe opt::PayoffButterfly	100
6.22.1 Description détaillée	101
6.22.2 Documentation des constructeurs et destructeur	101
6.22.2.1 PayoffButterfly()	101
6.22.3 Documentation des fonctions membres	101
6.22.3.1 operator()()	101
6.23 Référence de la classe opt::PayoffCall	102
6.23.1 Description détaillée	103
6.23.2 Documentation des constructeurs et destructeur	103

6.23.2.1 PayoffCall()	103
6.23.3 Documentation des fonctions membres	104
6.23.3.1 operator()()	104
6.24 Référence de la classe opt::PayoffDigitCall	104
6.24.1 Description détaillée	105
6.24.2 Documentation des constructeurs et destructeur	105
6.24.2.1 PayoffDigitCall()	105
6.24.3 Documentation des fonctions membres	105
6.24.3.1 operator()()	105
6.25 Référence de la classe opt::PayoffDigitPut	106
6.25.1 Description détaillée	107
6.25.2 Documentation des constructeurs et destructeur	107
6.25.2.1 PayoffDigitPut()	107
6.25.3 Documentation des fonctions membres	107
6.25.3.1 operator()()	107
6.26 Référence de la classe opt::PayoffDoubleDigit	108
6.26.1 Description détaillée	109
6.26.2 Documentation des constructeurs et destructeur	109
6.26.2.1 PayoffDoubleDigit()	109
6.26.3 Documentation des fonctions membres	110
6.26.3.1 operator()()	110
6.27 Référence de la classe opt::PayoffPut	110
6.27.1 Description détaillée	111
6.27.2 Documentation des constructeurs et destructeur	111
6.27.2.1 PayoffPut()	111
6.27.3 Documentation des fonctions membres	111
6.27.3.1 operator()()	111
6.28 Référence de la classe opt::PayoffStrangle	112
6.28.1 Description détaillée	113
6.28.2 Documentation des constructeurs et destructeur	113
6.28.2.1 PayoffStrangle()	113
6.28.3 Documentation des fonctions membres	113
6.28.3.1 operator()()	113
6.29 Référence de la classe pde::Volatility	114
6.29.1 Description détaillée	115
6.29.2 Documentation des fonctions membres	115
6.29.2.1 operator()()	115
7 Documentation des fichiers	117
7.1 Référence du fichier src/Aggregator.h	117
7.2 Aggregator.h	118
7.3 Référence du fichier src/American.h	119

7.4 American.h	120
7.5 Référence du fichier src/Asian.h	122
7.6 Asian.h	123
7.7 Référence du fichier src/CNMethod.h	125
7.7.1 Documentation des macros	126
7.7.1.1 CNMETHOD_H	126
7.8 CNMethod.h	127
7.9 Référence du fichier src/Diffusion.h	127
7.10 Diffusion.h	129
7.11 Référence du fichier src/dllmain.cpp	130
7.11.1 Documentation des fonctions	130
7.11.1.1 DllMain()	130
7.12 Référence du fichier src/European.h	130
7.13 European.h	132
7.14 Référence du fichier src/Exports.cpp	133
7.14.1 Documentation des macros	142
7.14.1.1 SAFE_DOUBLE	142
7.14.1.2 SAFE_VARIANT	143
7.14.2 Documentation des définitions de type	143
7.14.2.1 DiffusionBearBS	143
7.14.2.2 DiffusionBearVL	143
7.14.2.3 DiffusionBullBS	143
7.14.2.4 DiffusionBullVL	144
7.14.2.5 DiffusionButterflyBS	144
7.14.2.6 DiffusionButterflyVL	144
7.14.2.7 DiffusionDigitCallBS	144
7.14.2.8 DiffusionDigitCallVL	144
7.14.2.9 DiffusionDigitPutBS	144
7.14.2.10 DiffusionDigitPutVL	144
7.14.2.11 DiffusionDoubleDigitBS	144
7.14.2.12 DiffusionDoubleDigitVL	144
7.14.2.13 DiffusionPutBS	144
7.14.2.14 DiffusionPutVL	145
7.14.2.15 DiffusionStrangleBS	145
7.14.2.16 DiffusionStrangleVL	145
7.14.3 Documentation des fonctions	145
7.14.3.1 __declspec()	145
7.14.3.2 AsianCallArithmetic()	151
7.14.3.3 AsianCallGeometric()	151
7.14.3.4 AsianCallLookMax()	152
7.14.3.5 AsianPutArithmetic()	152
7.14.3.6 AsianPutGeometric()	153

7.14.3.7 AsianPutLookMin()	153
7.14.3.8 crr::American< opt::PayoffCall >()	154
7.14.3.9 crr::American< opt::PayoffPut >()	154
7.14.3.10 crr::European< opt::PayoffBear >()	154
7.14.3.11 crr::European< opt::PayoffBull >()	154
7.14.3.12 crr::European< opt::PayoffButterfly >()	154
7.14.3.13 crr::European< opt::PayoffCall >()	155
7.14.3.14 crr::European< opt::PayoffDigitCall >()	155
7.14.3.15 crr::European< opt::PayoffDigitPut >()	155
7.14.3.16 crr::European< opt::PayoffDoubleDigit >()	155
7.14.3.17 crr::European< opt::PayoffPut >()	155
7.14.3.18 crr::European< opt::PayoffStrangle >()	156
7.14.3.19 delta()	156
7.14.3.20 makeVariantFromArray()	156
7.14.3.21 reportError()	157
7.14.3.22 SAFE_DOUBLE() [1/43]	158
7.14.3.23 SAFE_DOUBLE() [2/43]	158
7.14.3.24 SAFE_DOUBLE() [3/43]	159
7.14.3.25 SAFE_DOUBLE() [4/43]	159
7.14.3.26 SAFE_DOUBLE() [5/43]	159
7.14.3.27 SAFE_DOUBLE() [6/43]	160
7.14.3.28 SAFE_DOUBLE() [7/43]	160
7.14.3.29 SAFE_DOUBLE() [8/43]	161
7.14.3.30 SAFE_DOUBLE() [9/43]	161
7.14.3.31 SAFE_DOUBLE() [10/43]	161
7.14.3.32 SAFE_DOUBLE() [11/43]	162
7.14.3.33 SAFE_DOUBLE() [12/43]	162
7.14.3.34 SAFE_DOUBLE() [13/43]	162
7.14.3.35 SAFE_DOUBLE() [14/43]	163
7.14.3.36 SAFE_DOUBLE() [15/43]	163
7.14.3.37 SAFE_DOUBLE() [16/43]	163
7.14.3.38 SAFE_DOUBLE() [17/43]	163
7.14.3.39 SAFE_DOUBLE() [18/43]	164
7.14.3.40 SAFE_DOUBLE() [19/43]	164
7.14.3.41 SAFE_DOUBLE() [20/43]	164
7.14.3.42 SAFE_DOUBLE() [21/43]	165
7.14.3.43 SAFE_DOUBLE() [22/43]	165
7.14.3.44 SAFE_DOUBLE() [23/43]	165
7.14.3.45 SAFE_DOUBLE() [24/43]	166
7.14.3.46 SAFE_DOUBLE() [25/43]	166
7.14.3.47 SAFE_DOUBLE() [26/43]	166
7.14.3.48 SAFE_DOUBLE() [27/43]	167

7.14.3.49 SAFE_DOUBLE() [28/43]	167
7.14.3.50 SAFE_DOUBLE() [29/43]	167
7.14.3.51 SAFE_DOUBLE() [30/43]	168
7.14.3.52 SAFE_DOUBLE() [31/43]	168
7.14.3.53 SAFE_DOUBLE() [32/43]	169
7.14.3.54 SAFE_DOUBLE() [33/43]	169
7.14.3.55 SAFE_DOUBLE() [34/43]	169
7.14.3.56 SAFE_DOUBLE() [35/43]	170
7.14.3.57 SAFE_DOUBLE() [36/43]	170
7.14.3.58 SAFE_DOUBLE() [37/43]	171
7.14.3.59 SAFE_DOUBLE() [38/43]	171
7.14.3.60 SAFE_DOUBLE() [39/43]	171
7.14.3.61 SAFE_DOUBLE() [40/43]	171
7.14.3.62 SAFE_DOUBLE() [41/43]	171
7.14.3.63 SAFE_DOUBLE() [42/43]	172
7.14.3.64 SAFE_DOUBLE() [43/43]	172
7.14.3.65 SAFE_VARIANT() [1/47]	172
7.14.3.66 SAFE_VARIANT() [2/47]	173
7.14.3.67 SAFE_VARIANT() [3/47]	173
7.14.3.68 SAFE_VARIANT() [4/47]	173
7.14.3.69 SAFE_VARIANT() [5/47]	174
7.14.3.70 SAFE_VARIANT() [6/47]	174
7.14.3.71 SAFE_VARIANT() [7/47]	175
7.14.3.72 SAFE_VARIANT() [8/47]	175
7.14.3.73 SAFE_VARIANT() [9/47]	176
7.14.3.74 SAFE_VARIANT() [10/47]	176
7.14.3.75 SAFE_VARIANT() [11/47]	177
7.14.3.76 SAFE_VARIANT() [12/47]	177
7.14.3.77 SAFE_VARIANT() [13/47]	178
7.14.3.78 SAFE_VARIANT() [14/47]	178
7.14.3.79 SAFE_VARIANT() [15/47]	179
7.14.3.80 SAFE_VARIANT() [16/47]	179
7.14.3.81 SAFE_VARIANT() [17/47]	180
7.14.3.82 SAFE_VARIANT() [18/47]	180
7.14.3.83 SAFE_VARIANT() [19/47]	181
7.14.3.84 SAFE_VARIANT() [20/47]	181
7.14.3.85 SAFE_VARIANT() [21/47]	182
7.14.3.86 SAFE_VARIANT() [22/47]	182
7.14.3.87 SAFE_VARIANT() [23/47]	182
7.14.3.88 SAFE_VARIANT() [24/47]	183
7.14.3.89 SAFE_VARIANT() [25/47]	183
7.14.3.90 SAFE_VARIANT() [26/47]	184

7.14.3.91 SAFE_VARIANT() [27/47]	184
7.14.3.92 SAFE_VARIANT() [28/47]	184
7.14.3.93 SAFE_VARIANT() [29/47]	185
7.14.3.94 SAFE_VARIANT() [30/47]	185
7.14.3.95 SAFE_VARIANT() [31/47]	186
7.14.3.96 SAFE_VARIANT() [32/47]	186
7.14.3.97 SAFE_VARIANT() [33/47]	186
7.14.3.98 SAFE_VARIANT() [34/47]	187
7.14.3.99 SAFE_VARIANT() [35/47]	187
7.14.3.100 SAFE_VARIANT() [36/47]	188
7.14.3.101 SAFE_VARIANT() [37/47]	188
7.14.3.102 SAFE_VARIANT() [38/47]	188
7.14.3.103 SAFE_VARIANT() [39/47]	189
7.14.3.104 SAFE_VARIANT() [40/47]	189
7.14.3.105 SAFE_VARIANT() [41/47]	190
7.14.3.106 SAFE_VARIANT() [42/47]	190
7.14.3.107 SAFE_VARIANT() [43/47]	190
7.14.3.108 SAFE_VARIANT() [44/47]	191
7.14.3.109 SAFE_VARIANT() [45/47]	191
7.14.3.110 SAFE_VARIANT() [46/47]	192
7.14.3.111 SAFE_VARIANT() [47/47]	192
7.14.3.112 SolvePDE()	192
7.14.3.113 solver()	193
7.14.3.114 toVariant() [1/4]	194
7.14.3.115 toVariant() [2/4]	195
7.14.3.116 toVariant() [3/4]	195
7.14.3.117 toVariant() [4/4]	195
7.14.4 Documentation des variables	196
7.14.4.1 alfa	196
7.14.4.2 beta	196
7.14.4.3 G	196
7.14.4.4 imax	196
7.14.4.5 jmax	196
7.14.4.6 K	196
7.14.4.7 K1	196
7.14.4.8 K2	196
7.14.4.9 N	196
7.14.4.10 R	196
7.14.4.11 S	197
7.14.4.12 S0	197
7.14.4.13 sigma	197
7.14.4.14 Smax	197

7.14.4.15 Smin	197
7.14.4.16 T	197
7.14.4.17 t	197
7.15 Référence du fichier src/Exports.h	197
7.15.1 Documentation des fonctions	199
7.15.1.1 __declspec()	199
7.15.2 Documentation des variables	205
7.15.2.1 alfa	205
7.15.2.2 beta	205
7.15.2.3 imax	205
7.15.2.4 jmax	205
7.15.2.5 K	205
7.15.2.6 K1	205
7.15.2.7 K2	205
7.15.2.8 N	205
7.15.2.9 R	205
7.15.2.10 S	205
7.15.2.11 sigma	206
7.15.2.12 Smax	206
7.15.2.13 Smin	206
7.15.2.14 T	206
7.16 Exports.h	206
7.17 Référence du fichier src/FDMethod.h	216
7.18 FDMethod.h	217
7.19 Référence du fichier src/framework.h	219
7.19.1 Documentation des macros	220
7.19.1.1 WIN32_LEAN_AND_MEAN	220
7.20 framework.h	220
7.21 Référence du fichier src/ImplicitScheme.h	220
7.22 ImplicitScheme.h	221
7.23 Référence du fichier src/Option.cpp	222
7.24 Référence du fichier src/Option.h	223
7.25 Option.h	224
7.26 Référence du fichier src/ParabPDE.h	225
7.27 ParabPDE.h	226
7.28 Référence du fichier src/Payoff.cpp	227
7.29 Référence du fichier src/Payoff.h	228
7.30 Payoff.h	229
7.31 Référence du fichier src/pch.cpp	231
7.32 Référence du fichier src/pch.h	231
7.33 pch.h	232
7.34 Référence du fichier src/Volatility.cpp	232

7.35 Référence du fichier src/Volatility.h	233
7.36 Volatility.h	235
Index	237

Chapitre 1

Index des espaces de nommage

1.1 Liste des espaces de nommage

Liste de tous les espaces de nommage avec une brève description:

crr	9
opt	10
pde	10

Chapitre 2

Index hiérarchique

2.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

crr::Aggregator	13
crr::Arithmetic	21
crr::Geometric	61
crr::LookMax	78
crr::LookMin	79
pde::FDMETHOD< TPDE >	51
pde::ImplicitScheme< TPDE >	65
pde::CNMethod< TPDE >	33
pde::FDMETHOD< TPDE >::Grid	63
crr::Option::HedgingStrategy	64
crr::Option	81
crr::American< TPAYOFF >	14
crr::Asian< TPAYOFF, TAggregator >	23
crr::European< TPAYOFF >	46
pde::ParabPDE	88
pde::Diffusion< opt::PayoffPut, pde::BSVol >	41
pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >	41
pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >	41
pde::Diffusion< opt::PayoffDoubleDigit, pde::BSVol >	41
pde::Diffusion< opt::PayoffBull, pde::BSVol >	41
pde::Diffusion< opt::PayoffBear, pde::BSVol >	41
pde::Diffusion< opt::PayoffStrangle, pde::BSVol >	41
pde::Diffusion< opt::PayoffButterfly, pde::BSVol >	41
pde::Diffusion< opt::PayoffPut, pde::LocalVol >	41
pde::Diffusion< opt::PayoffDigitCall, pde::LocalVol >	41
pde::Diffusion< opt::PayoffDigitPut, pde::LocalVol >	41
pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >	41
pde::Diffusion< opt::PayoffBull, pde::LocalVol >	41
pde::Diffusion< opt::PayoffBear, pde::LocalVol >	41
pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >	41
pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >	41
pde::Diffusion< TPAYOFF, TVOL >	41
opt::Payoff	95
opt::PayoffBear	96

opt::PayoffBull	98
opt::PayoffButterfly	100
opt::PayoffCall	102
opt::PayoffDigitCall	104
opt::PayoffDigitPut	106
opt::PayoffDoubleDigit	108
opt::PayoffPut	110
opt::PayoffStrangle	112
pde::Volatility	114
pde::BSVol	30
pde::LocalVol	75

Chapitre 3

Index des classes

3.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

crr::Aggregator	Interface abstraite pour l'agrégation des prix path-dépendants	13
crr::American< TPayoff >	Option américaine CRR avec couverture via décomposition de Doob	14
crr::Arithmetic	Agrégateur pour la moyenne arithmétique	21
crr::Asian< TPayoff, TAggregator >	Options path-dépendante CRR	23
pde::BSVol	Volatilité dans le modèle de Black-Scholes	30
pde::CNMethod< TPDE >	Méthode de Crank–Nicolson pour le schéma implicite aux différences finies	33
pde::Diffusion< TPayoff, TVol >	EDP d'un modèle de diffusion	41
crr::European< TPayoff >	Option européenne CRR	46
pde::FDMMethod< TPDE >	Méthodes aux différences finies pour la résolution numérique d'EDP	51
crr::Geometric	Agrégateur pour la moyenne géométrique	61
pde::FDMMethod< TPDE >::Grid	Grille : prix et delta	63
crr::Option::HedgingStrategy	Stratégie de couverture : delta et obligation	64
pde::ImplicitScheme< TPDE >	Schéma implicite aux différences finies	65
pde::LocalVol	Volatilité dans un modèle à volatilité locale	75
crr::LookMax	Agrégateur pour le maximum	78
crr::LookMin	Agrégateur pour le minimum	79
crr::Option	Classe de base Option : calcul des paramètres du modèle	81
pde::ParabPDE	Interface pour EDP paraboliques de type général	88

opt::Payoff	
Classe abstraite représentant le payoff d'une option	95
opt::PayoffBear	
Payoff d'une option bear spread	96
opt::PayoffBull	
Payoff d'une option bull spread	98
opt::PayoffButterfly	
Payoff d'une option butterfly	100
opt::PayoffCall	
Payoff d'une option d'achat européenne (call)	102
opt::PayoffDigitCall	
Payoff d'un call digital	104
opt::PayoffDigitPut	
Payoff d'un put digital	106
opt::PayoffDoubleDigit	
Payoff d'une option double-digital	108
opt::PayoffPut	
Payoff d'une option de vente européenne (put)	110
opt::PayoffStrangle	
Payoff d'une option strangle	112
pde::Volatility	
Classe abstraite représentant la volatilité du sous-jacent	114

Chapitre 4

Index des fichiers

4.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

src/Aggregator.h	117
src/American.h	119
src/Asian.h	122
src/CNMethod.h	125
src/Diffusion.h	127
src/dllmain.cpp	130
src/European.h	130
src/Exports.cpp	133
src/Exports.h	197
src/FDMethod.h	216
src/framework.h	219
src/ImplicitScheme.h	220
src/Option.cpp	222
src/Option.h	223
src/ParabPDE.h	225
src/Payoff.cpp	227
src/Payoff.h	228
src/pch.cpp	231
src/pch.h	231
src/Volatility.cpp	232
src/Volatility.h	233

Chapitre 5

Documentation des espaces de nommage

5.1 Référence de l'espace de nommage crr

Classes

- class [Aggregator](#)

Interface abstraite pour l'agrégation des prix path-dépendants.

- class [American](#)

Option américaine CRR avec couverture via décomposition de Doob.

- class [Arithmetic](#)

Agrégateur pour la moyenne arithmétique.

- class [Asian](#)

Options path-dépendante CRR.

- class [European](#)

Option européenne CRR.

- class [Geometric](#)

Agrégateur pour la moyenne géométrique.

- class [LookMax](#)

Agrégateur pour le maximum.

- class [LookMin](#)

Agrégateur pour le minimum.

- class [Option](#)

Classe de base [Option](#) : calcul des paramètres du modèle.

5.2 Référence de l'espace de nommage opt

Classes

- class [Payoff](#)

Classe abstraite représentant le payoff d'une option.

- class [PayoffBear](#)

Payoff d'une option bear spread.

- class [PayoffBull](#)

Payoff d'une option bull spread.

- class [PayoffButterfly](#)

Payoff d'une option butterfly.

- class [PayoffCall](#)

Payoff d'une option d'achat européenne (call).

- class [PayoffDigitCall](#)

Payoff d'un call digital.

- class [PayoffDigitPut](#)

Payoff d'un put digital.

- class [PayoffDoubleDigit](#)

Payoff d'une option double-digital.

- class [PayoffPut](#)

Payoff d'une option de vente européenne (put).

- class [PayoffStrangle](#)

Payoff d'une option strangle.

5.3 Référence de l'espace de nommage pde

Classes

- class [BSVol](#)

Volatilité dans le modèle de Black-Scholes.

- class [CNMethod](#)

Méthode de Crank–Nicolson pour le schéma implicite aux différences finies.

- class [Diffusion](#)

EDP d'un modèle de diffusion.

- class [FDMethod](#)

Méthodes aux différences finies pour la résolution numérique d'EDP.

- class [ImplicitScheme](#)

Schéma implicite aux différences finies.

- class [LocalVol](#)

Volatilité dans un modèle à volatilité locale.

- class [ParabPDE](#)

Interface pour EDP paraboliques de type général.

- class [Volatility](#)

Classe abstraite représentant la volatilité du sous-jacent.

Chapitre 6

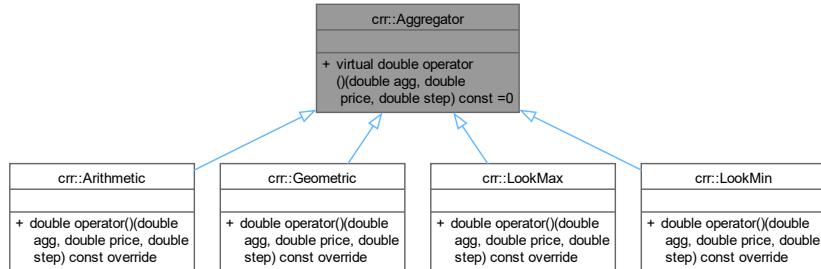
Documentation des classes

6.1 Référence de la classe crr::Aggregator

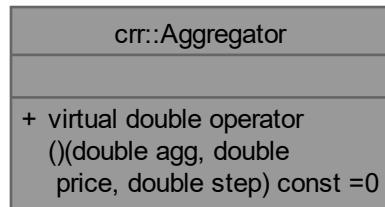
Interface abstraite pour l'agrégation des prix path-dépendants.

```
#include <Aggregator.h>
```

Graphe d'héritage de crr::Aggregator:



Graphe de collaboration de crr::Aggregator:



Fonctions membres publiques

— virtual double [operator\(\)](#) (double agg, double price, double step) const =0

Calcule la nouvelle valeur agrégée.

6.1.1 Description détaillée

Interface abstraite pour l'agrégation des prix path-dépendants.

6.1.2 Documentation des fonctions membres

6.1.2.1 operator()()

```
virtual double crr::Aggregator::operator() (
    double agg,
    double price,
    double step) const [pure virtual]
```

Calcule la nouvelle valeur agrégée.

Paramètres

<i>agg</i>	Valeur agrégée jusqu'à l'étape précédente.
<i>price</i>	Prix courant du sous-jacent.
<i>step</i>	Numéro de l'étape.

Renvoie

Valeur agrégée mise à jour.

Implémenté dans [crr::Arithmetic](#), [crr::Geometric](#), [crr::LookMax](#), et [crr::LookMin](#).

La documentation de cette classe a été générée à partir du fichier suivant :

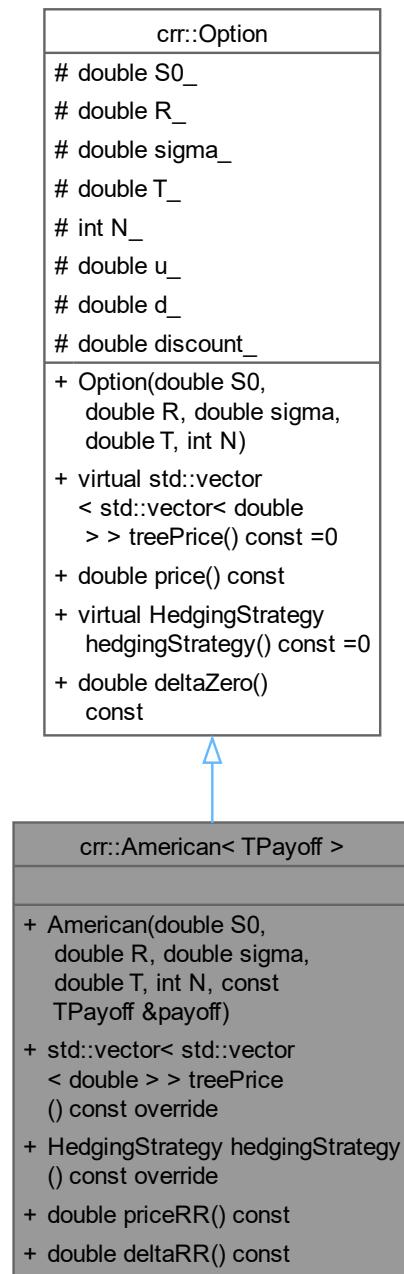
— [src/Aggregator.h](#)

6.2 Référence du modèle de la classe `crr::American< TPayoff >`

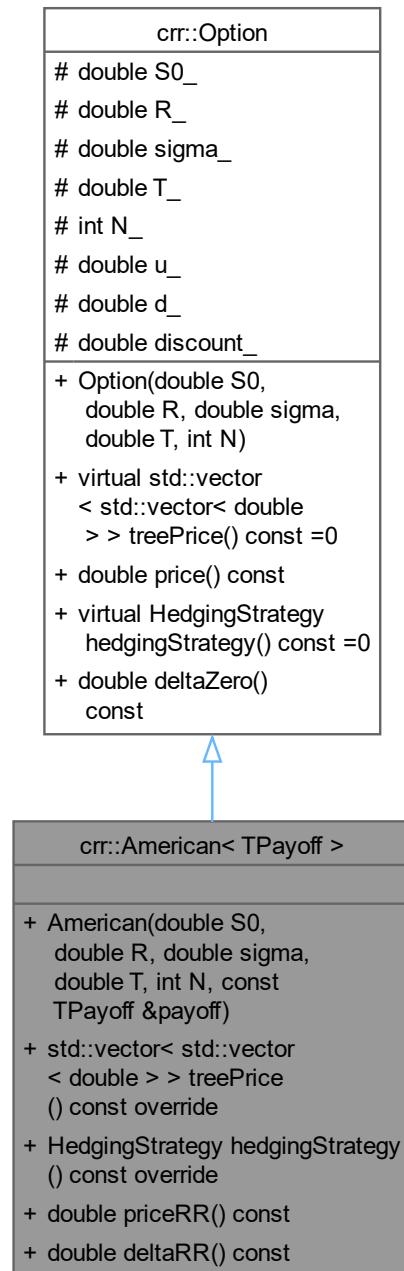
Option américaine CRR avec couverture via décomposition de Doob.

```
#include <American.h>
```

Graphe d'héritage de crr::American< TPayoff >:



Graphe de collaboration de `crr::American< TPayoff >`:



Fonctions membres publiques

- `American` (double **S0**, double **R**, double **sigma**, double **T**, int **N**, const `TPayoff &payoff`)
- `std::vector< std::vector< double > > treePrice () const override`

Arbre des prix de l'option selon le modèle CRR.

- `HedgingStrategy hedgingStrategy () const`
Calcule la couverture optimale (CRR).
- `double priceRR () const`
Prix asymptotique de l'option (extrapolation répétée de Richardson).
- `double deltaRR () const`
Delta asymptotique (bump-and-reprice).

Fonctions membres publiques hérités de `crr::Option`

- `Option (double S0, double R, double sigma, double T, int N)`
Constructeur : initialise les paramètres de l'option pour le modèle CRR.
- `double price () const`
Prix initial de l'option.
- `double deltaZero () const`
Delta initial.

Membres hérités additionnels

Attributs protégés hérités de `crr::Option`

- `double S0_`
- `double R_`
- `double sigma_`
- `double T_`
Paramètres initiaux.
- `int N_`
Nombre de pas.
- `double u_`
- `double d_`
- `double discount_`
Paramètres par pas.

6.2.1 Description détaillée

```
template<typename TPayoff>
class crr::American< TPayoff >
```

Option américaine CRR avec couverture via décomposition de Doob.

Paramètres du template

<i>TPayoff</i>	Type de payoff.
----------------	-----------------

6.2.2 Documentation des constructeurs et destructeur

6.2.2.1 American()

```
template<typename TPayoff>
crr::American< TPayoff >::American (
    double s0,
    double R,
    double sigma,
    double T,
    int N,
    const TPayoff & payoff)
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.2.3 Documentation des fonctions membres

6.2.3.1 `deltaRR()`

```
template<typename TPayoff>
double crr::American< TPayoff >::deltaRR () const
```

Delta asymptotique (bump-and-reprice).

Renvoie

Valeur du delta à n = 0.

Voici le graphe d'appel pour cette fonction :



6.2.3.2 `hedgingStrategy()`

```
template<typename TPayoff>
Option::HedgingStrategy crr::American< TPayoff >::hedgingStrategy () const [override], [virtual]
```

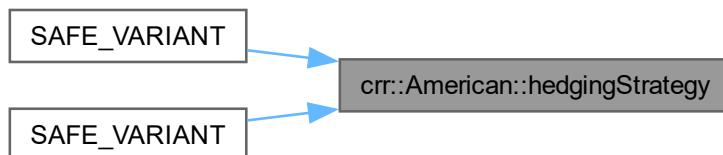
Calcule la couverture optimale (CRR).

Renvoie

`HedgingStrategy` contenant delta et bond par nœud.

Implémente `crr::Option`.

Voici le graphe des appels de cette fonction :



6.2.3.3 priceRR()

```
template<typename TPayoff>
double crr::American< TPayoff >::priceRR () const
```

Prix asymptotique de l'option (extrapolation répétée de Richardson).

Renvoie

Valeur de l'option à n = 0.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.2.3.4 treePrice()

```
template<typename TPayoff>
std::vector< std::vector< double > > crr::American< TPayoff >::treePrice () const [override],
[virtual]
```

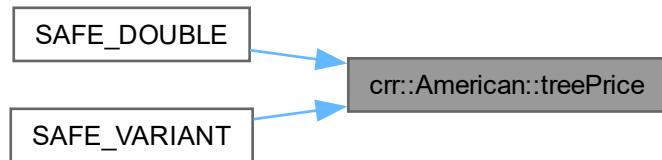
Arbre des prix de l'option selon le modèle CRR.

Renvoie

Matrice des valeurs aux nœuds.

Implémente [crr::Option](#).

Voici le graphe des appels de cette fonction :



La documentation de cette classe a été générée à partir du fichier suivant :

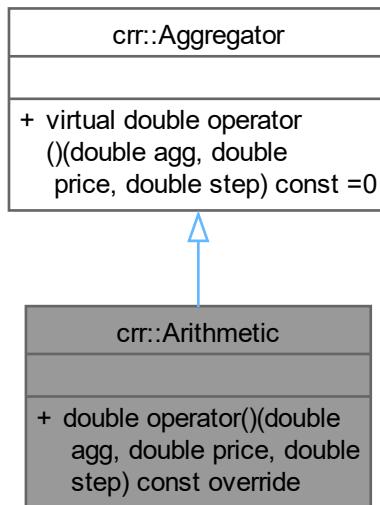
— [src/American.h](#)

6.3 Référence de la classe crr::Arithmetic

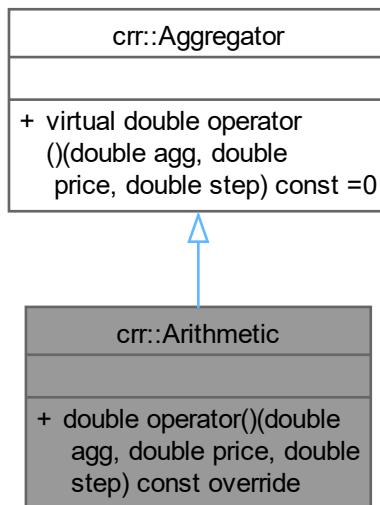
Agrégateur pour la moyenne arithmétique.

```
#include <Aggregator.h>
```

Graphe d'héritage de crr::Arithmetic:



Graphe de collaboration de crr::Arithmetic:



Fonctions membres publiques

— double **operator()** (double agg, double price, double step) const override

Calcule la nouvelle valeur agrégée.

6.3.1 Description détaillée

Agrégateur pour la moyenne arithmétique.

6.3.2 Documentation des fonctions membres

6.3.2.1 **operator()()**

```
double crr::Arithmetic::operator() (
    double agg,
    double price,
    double step) const [inline], [override], [virtual]
```

Calcule la nouvelle valeur agrégée.

Paramètres

<i>agg</i>	Valeur agrégée jusqu'à l'étape précédente.
<i>price</i>	Prix courant du sous-jacent.
<i>step</i>	Numéro de l'étape.

Renvoie

Valeur agrégée mise à jour.

Implémente [crr::Aggregator](#).

La documentation de cette classe a été générée à partir du fichier suivant :

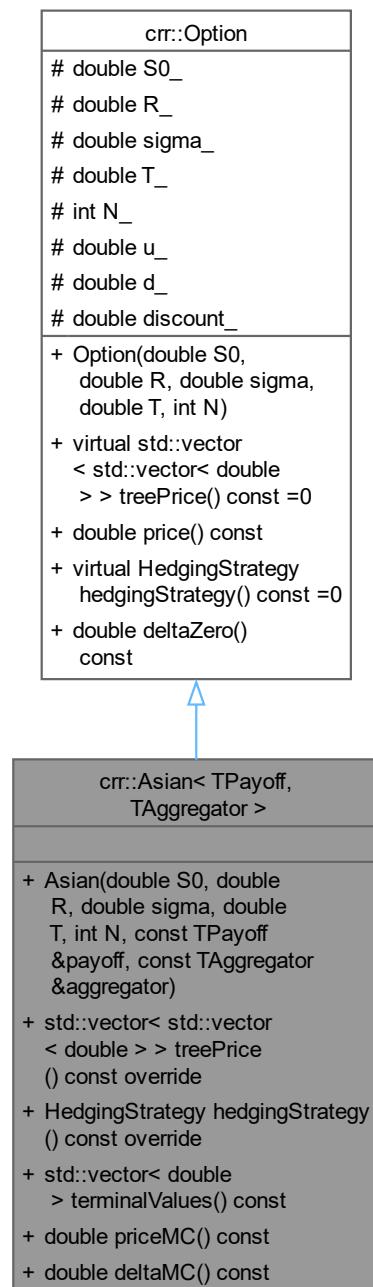
— [src/Aggregator.h](#)

6.4 Référence du modèle de la classe `crr::Asian< TPayoff, TAggregator >`

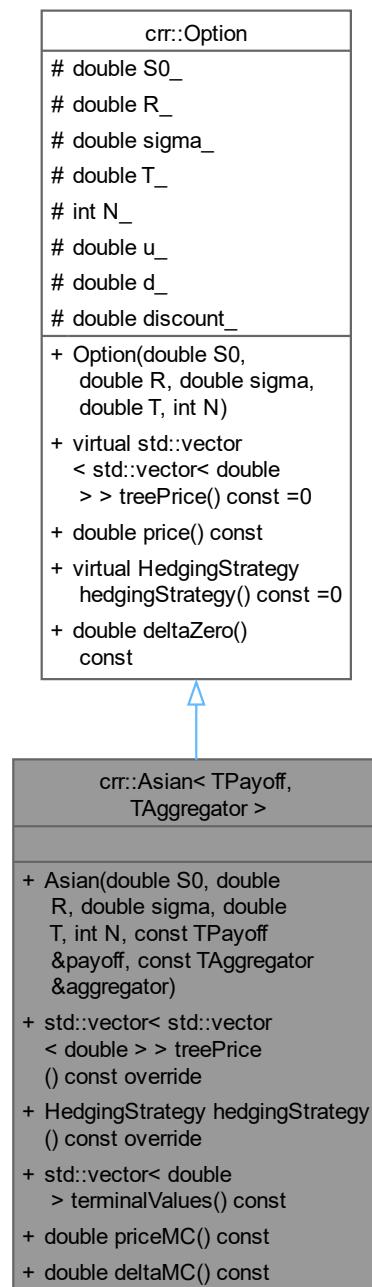
Options path-dépendante CRR.

```
#include <Asian.h>
```

Graphe d'héritage de `crr::Asian< TPayoff, TAggregator >`:



Graphe de collaboration de crr::Asian< TPayoff, TAggregator >:



Fonctions membres publiques

- **Asian** (double **S0**, double **R**, double **sigma**, double **T**, int **N**, const **TPayoff** &**payoff**, const **TAggregator** &**aggregator**)
- **std::vector< std::vector< double >> treePrice () const override**

Arbre des prix de l'option selon le modèle CRR.

- `HedgingStrategy hedgingStrategy () const override`

Calcule la couverture optimale (CRR).

- `std::vector< double > terminalValues () const`

Valeurs terminales de l'option path-dépendante.

- `double priceMC () const`

Prix asymptotique de l'option (méthode Monte Carlo).

- `double deltaMC () const`

Delta asymptotique (méthode bump-and-reprice).

Fonctions membres publiques hérités de `crr::Option`

- `Option (double S0, double R, double sigma, double T, int N)`

Constructeur : initialise les paramètres de l'option pour le modèle CRR.

- `double price () const`

Prix initial de l'option.

- `double deltaZero () const`

Delta initial.

Membres hérités additionnels

Attributs protégés hérités de `crr::Option`

- `double S0_`

- `double R_`

- `double sigma_`

- `double T_`

Paramètres initiaux.

- `int N_`

Nombre de pas.

- `double u_`

- `double d_`

- `double discount_`

Paramètres par pas.

6.4.1 Description détaillée

```
template<typename TPayoff, typename TAggregator>
class crr::Asian< TPayoff, TAggregator >
```

Options path-dépendante CRR.

Paramètres du template

<i>TPayoff</i>	Type de payoff.
<i>TAggregator</i>	Type d'agrégateur.

6.4.2 Documentation des constructeurs et destructeur

6.4.2.1 Asian()

```
template<typename TPayoff, typename TAggregator>
crr::Asian< TPayoff, TAggregator >::Asian (
    double S0,
    double R,
    double sigma,
    double T,
    int N,
    const TPayoff & payoff,
    const TAggregator & aggregator)
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.4.3 Documentation des fonctions membres

6.4.3.1 deltaMC()

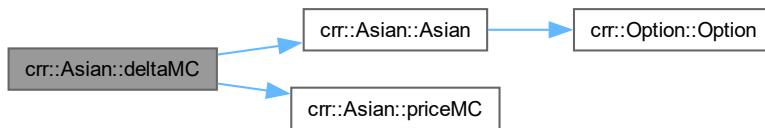
```
template<typename TPayoff, typename TAggregator>
double crr::Asian< TPayoff, TAggregator >::deltaMC () const
```

Delta asymptotique (méthode bump-and-reprice).

Renvoie

Valeur du delta à n = 0.

Voici le graphe d'appel pour cette fonction :



6.4.3.2 hedgingStrategy()

```
template<typename TPayoff, typename TAggregator>
Option::HedgingStrategy crr::Asian< TPayoff, TAggregator >::hedgingStrategy () const [override], [virtual]
```

Calcule la couverture optimale (CRR).

Renvoie

`HedgingStrategy` contenant delta et bond par nœud.

Implémente `crr::Option`.

Voici le graphe d'appel pour cette fonction :



6.4.3.3 `priceMC()`

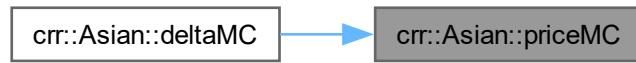
```
template<typename TPayoff, typename TAggregator>
double crr::Asian< TPayoff, TAggregator >::priceMC () const
```

Prix asymptotique de l'option (méthode Monte Carlo).

Renvoie

Valeur de l'option à $n = 0$.

Voici le graphe des appelants de cette fonction :



6.4.3.4 `terminalValues()`

```
template<typename TPayoff, typename TAggregator>
std::vector< double > crr::Asian< TPayoff, TAggregator >::terminalValues () const
```

Valeurs terminales de l'option path-dépendante.

Renvoie

Vecteur des payoffs à l'échéance pour chaque trajectoire du sous-jacent.

Voici le graphe des appelants de cette fonction :



6.4.3.5 treePrice()

```
template<typename TPayoff, typename TAggregator>
std::vector< std::vector< double > > crr::Asian< TPayoff, TAggregator >::treePrice () const
[override], [virtual]
```

Arbre des prix de l'option selon le modèle CRR.

Renvoie

Matrice des valeurs aux nœuds.

Implémente [crr::Option](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



La documentation de cette classe a été générée à partir du fichier suivant :

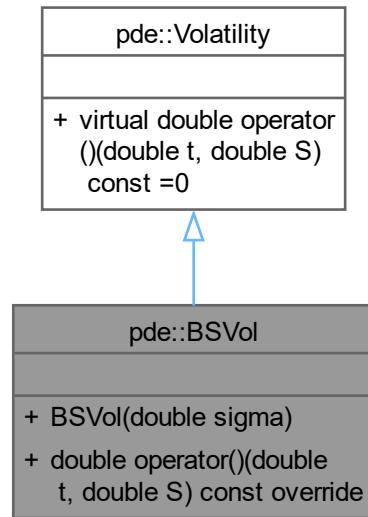
— [src/Asian.h](#)

6.5 Référence de la classe pde::BSVol

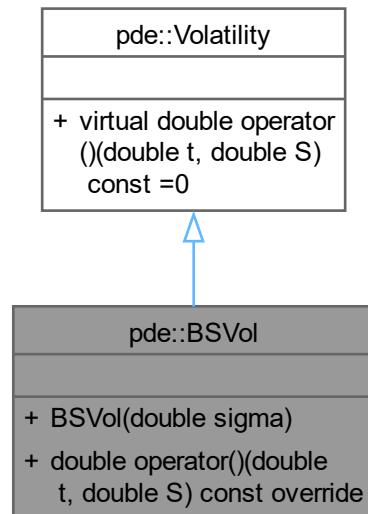
Volatilité dans le modèle de Black-Scholes.

```
#include <Volatility.h>
```

Graphe d'héritage de pde::BSVol:



Graphe de collaboration de pde::BSVol:



Fonctions membres publiques

- `BSVol (double sigma)`

- double **operator()** (double **t**, double **S**) const override

Calcule la valeur de la volatilité pour un prix S du sous-jacent à l'instant t.

6.5.1 Description détaillée

Volatilité dans le modèle de Black-Scholes.

6.5.2 Documentation des constructeurs et destructeur

6.5.2.1 BSVol()

```
pde::BSVol::BSVol (
    double sigma)
```

Paramètres

<i>sigma</i>	Valeur constante de la volatilité.
--------------	------------------------------------

6.5.3 Documentation des fonctions membres

6.5.3.1 operator()()

```
double pde::BSVol::operator() (
    double t,
    double S) const [override], [virtual]
```

Calcule la valeur de la volatilité pour un prix S du sous-jacent à l'instant t.

Paramètres

<i>t</i>	Temps.
<i>S</i>	Prix du sous-jacent.

Renvoie

Valeur de la volatilité.

Implémente [pde::Volatility](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

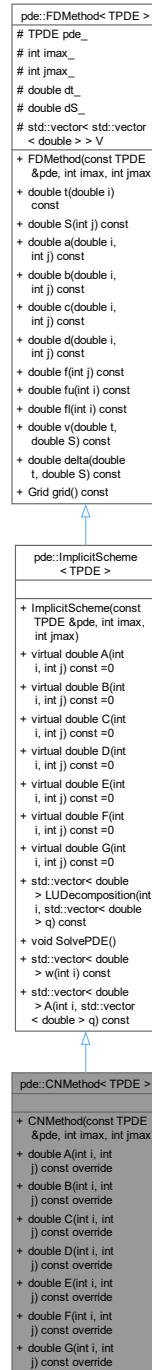
- src/[Volatility.h](#)
- src/[Volatility.cpp](#)

6.6 Référence du modèle de la classe pde::CNMethod< TPDE >

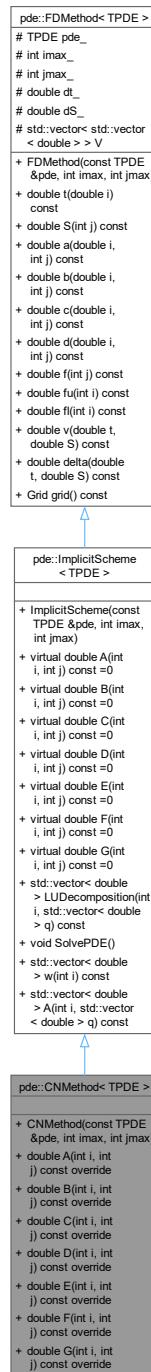
Méthode de Crank–Nicolson pour le schéma implicite aux différences finies.

```
#include <CNMethod.h>
```

Graphe d'héritage de pde::CNMethod< TPDE >:



Graphe de collaboration de pde::CNMethod< TPDE >:



Fonctions membres publiques

- `CNMethod (const TPDE &pde, int imax, int jmax)`
- `double A (int i, int j) const override`

Coefficients de l'équation discrétisée.

- double **B** (int i, int j) const override
- double **C** (int i, int j) const override
- double **D** (int i, int j) const override
- double **E** (int i, int j) const override
- double **F** (int i, int j) const override
- double **G** (int i, int j) const override

Fonctions membres publiques hérités de pde::ImplicitScheme< TPDE >

- **ImplicitScheme** (const TPDE &pde, int **imax**, int **jmax**)
- std::vector< double > **LUDecomposition** (int i, std::vector< double > q) const

Résolution d'un système tridiagonal par l'algorithme de Thomas.

- void **SolvePDE** ()

Calcul de la matrice de solution V.

- std::vector< double > **w** (int i) const
- std::vector< double > **A** (int i, std::vector< double > q) const

Fonctions membres publiques hérités de pde::FDMethod< TPDE >

- **FDMethod** (const TPDE &pde, int **imax**, int **jmax**)
- double **t** (double i) const
- double **S** (int j) const
- double **a** (double i, int j) const
- double **b** (double i, int j) const
- double **c** (double i, int j) const
- double **d** (double i, int j) const
- double **f** (int j) const
- double **fu** (int i) const
- double **fl** (int i) const
- double **v** (double **t**, double **S**) const

Prix de l'option par interpolation bilinéaire.

- double **delta** (double **t**, double **S**) const

Delta de l'option par schéma aux différences centrales.

— [Grid grid \(\) const](#)

Génère un maillage fixe de 11×11 points pour (t, S) .

Membres hérités additionnels

Attributs protégés hérités de [pde::FDMethod< TPDE >](#)

— [TPDE pde_](#)

— [int imax_](#)

— [int jmax_](#)

Nombre de pas en temps et en prix.

— [double dt_](#)

— [double dS_](#)

Pas en temps et en prix.

— [std::vector< std::vector< double > > V](#)

Matrice de solution.

6.6.1 Description détaillée

```
template<typename TPDE>
class pde::CNMethod< TPDE >
```

Méthode de Crank–Nicolson pour le schéma implicite aux différences finies.

Paramètres du template

<i>TPDE</i>	Type d'EDP.
-------------	-------------

6.6.2 Documentation des constructeurs et destructeur

6.6.2.1 CNMethod()

```
template<typename TPDE>
pde::CNMethod< TPDE >::CNMethod (
    const TPDE & pde,
    int imax,
    int jmax) [inline]
```

Voici le graphe d'appel pour cette fonction :



6.6.3 Documentation des fonctions membres

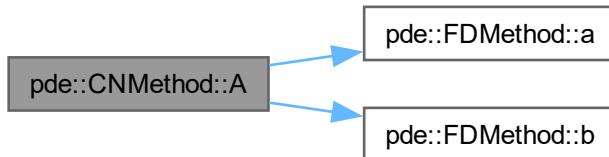
6.6.3.1 A()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::A (
    int i,
    int j) const [inline], [override], [virtual]
```

Coefficients de l'équation discrétisée.

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :

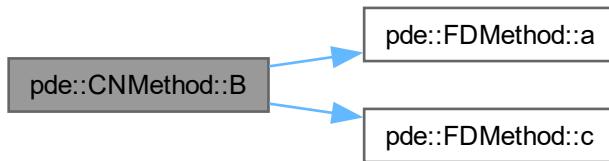


6.6.3.2 B()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::B (
    int i,
    int j) const [inline], [override], [virtual]
```

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :

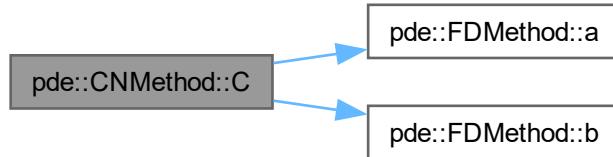


6.6.3.3 C()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::C (
    int i,
    int j) const [inline], [override], [virtual]
```

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.6.3.4 D()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::D (
    int i,
    int j) const [inline], [override], [virtual]
```

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :



6.6.3.5 E()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::E (
    int i,
    int j) const [inline], [override], [virtual]
```

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :

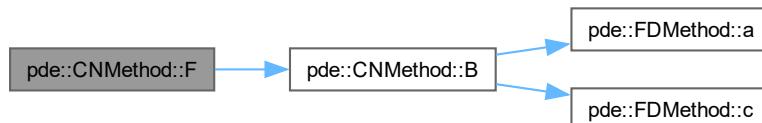


6.6.3.6 F()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::F (
    int i,
    int j) const [inline], [override], [virtual]
```

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :

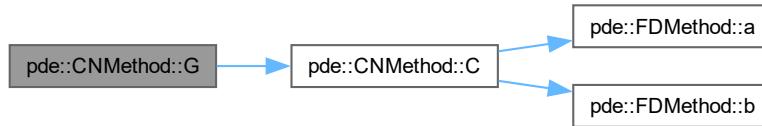


6.6.3.7 G()

```
template<typename TPDE>
double pde::CNMethod< TPDE >::G (
    int i,
    int j) const [inline], [override], [virtual]
```

Implémente [pde::ImplicitScheme< TPDE >](#).

Voici le graphe d'appel pour cette fonction :



La documentation de cette classe a été générée à partir du fichier suivant :

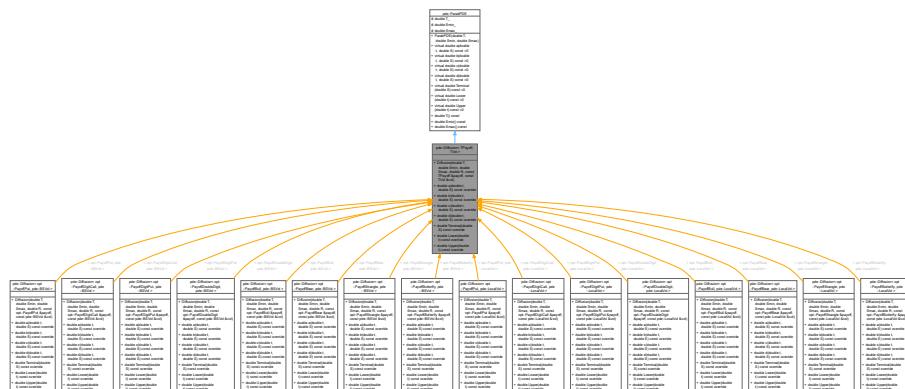
— [src/CNMethod.h](#)

6.7 Référence du modèle de la classe pde::Diffusion< TPayoff, TVol >

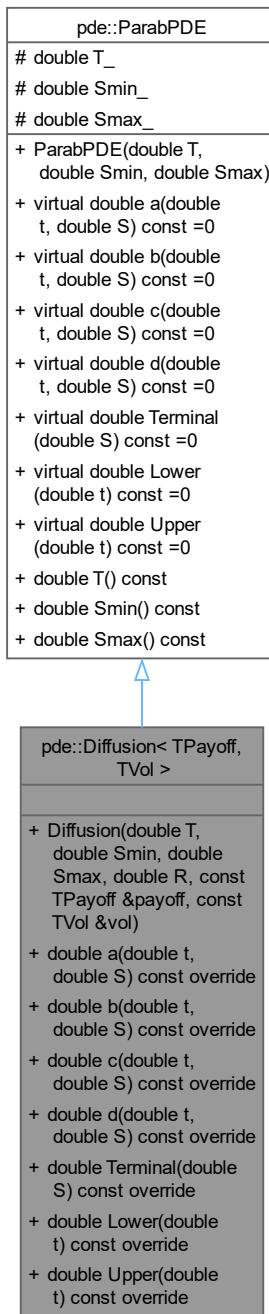
EDP d'un modèle de diffusion.

```
#include <Diffusion.h>
```

Graphe d'héritage de pde::Diffusion< TPayoff, TVol >:



Graphe de collaboration de pde::Diffusion< TPayoff, TVol >:



Fonctions membres publiques

- [Diffusion](#) (double **T**, double **Smin**, double **Smax**, double **R**, const **TPayoff** &**payoff**, const **TVol** &**vol**)
- [double a](#) (double **t**, double **S**) const override

Coefficients de l'équation différentielle partielle.

- double **b** (double **t**, double **S**) const override
- double **c** (double **t**, double **S**) const override
- double **d** (double **t**, double **S**) const override
- double **Terminal** (double **S**) const override

Terminal Boundary Condition.

- double **Lower** (double **t**) const override

Lower Boundary Condition.

- double **Upper** (double **t**) const override

Upper Boundary Condition.

Fonctions membres publiques hérités de pde::ParabPDE

- ParabPDE (double **T**, double **Smin**, double **Smax**)
- double **T** () const
- double **Smin** () const
- double **Smax** () const

Membres hérités additionnels

Attributs protégés hérités de pde::ParabPDE

- double **T_**
- double **Smin_**
- double **Smax_**

Domaine de l'EDP.

6.7.1 Description détaillée

```
template<typename TPayoff, typename TVol>
class pde::Diffusion< TPayoff, TVol >
```

EDP d'un modèle de diffusion.

Paramètres du template

<i>TPayoff</i>	Type de payoff.
<i>TVol</i>	Type de volatilité.

6.7.2 Documentation des constructeurs et destructeur

6.7.2.1 Diffusion()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::Diffusion (
    double T,
    double Smin,
    double Smax,
    double R,
    const TPayoff & payoff,
    const TVol & vol) [inline]
```

6.7.3 Documentation des fonctions membres

6.7.3.1 a()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::a (
    double t,
    double S) const [inline], [override], [virtual]
```

Coefficients de l'équation différentielle partielle.

Implémente [pde::ParabPDE](#).

6.7.3.2 b()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::b (
    double t,
    double S) const [inline], [override], [virtual]
```

Implémente [pde::ParabPDE](#).

6.7.3.3 c()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::c (
    double t,
    double S) const [inline], [override], [virtual]
```

Implémente [pde::ParabPDE](#).

6.7.3.4 d()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::d (
    double t,
    double S) const [inline], [override], [virtual]
```

Implémente [pde::ParabPDE](#).

6.7.3.5 Lower()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::Lower (
    double t) const [inline], [override], [virtual]
```

Lower Boundary Condition.

Renvoie

$v(t, S_{min})$.

Implémente [pde::ParabPDE](#).

6.7.3.6 Terminal()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::Terminal (
    double S) const [inline], [override], [virtual]
```

Terminal Boundary Condition.

Renvoie

$v(T, S)$.

Implémente [pde::ParabPDE](#).

6.7.3.7 Upper()

```
template<typename TPayoff, typename TVol>
double pde::Diffusion< TPayoff, TVol >::Upper (
    double t) const [inline], [override], [virtual]
```

Upper Boundary Condition.

Renvoie

$v(t, S_{max})$.

Implémente [pde::ParabPDE](#).

La documentation de cette classe a été générée à partir du fichier suivant :

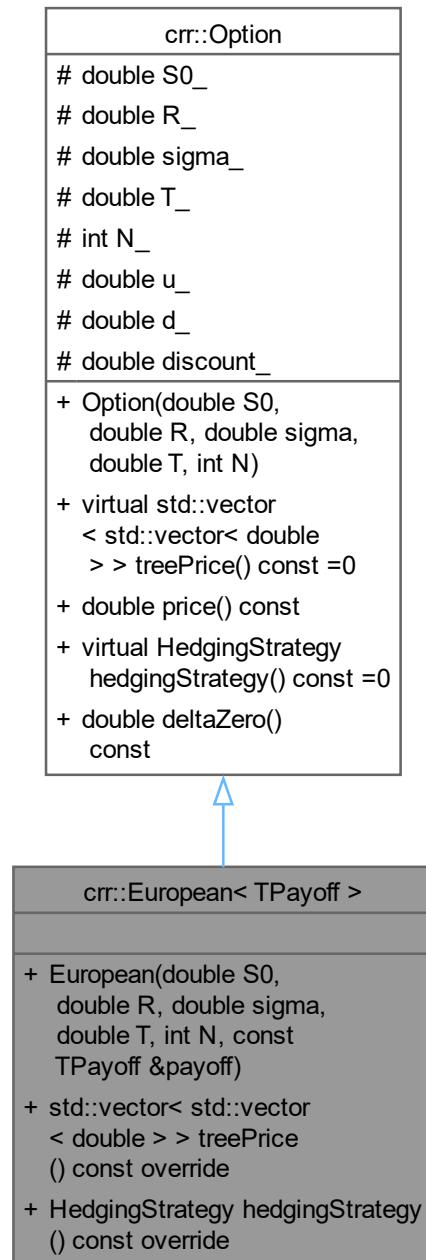
— src/[Diffusion.h](#)

6.8 Référence du modèle de la classe crr::European< TPayoff >

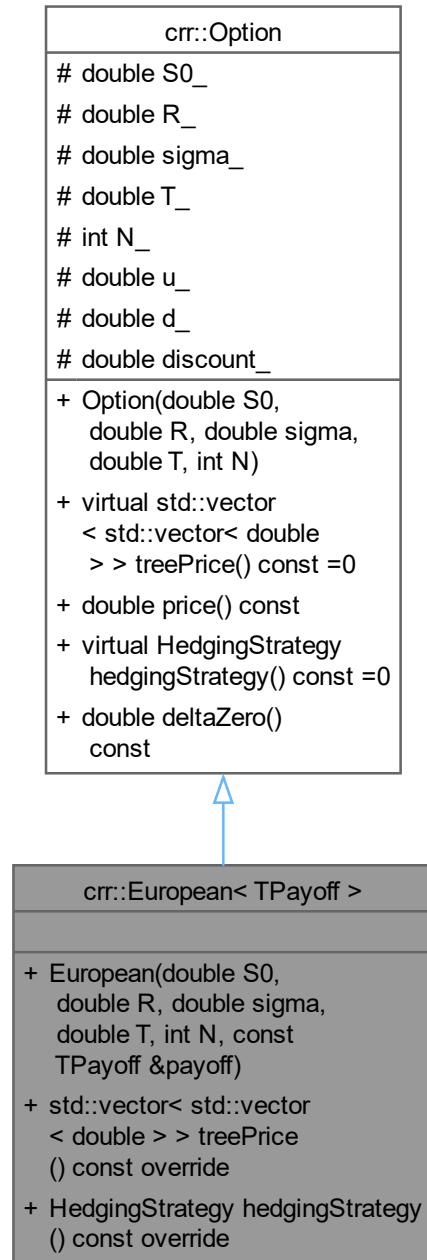
Option européenne CRR.

```
#include <European.h>
```

Graphe d'héritage de crr::European< TPayoff >:



Graphe de collaboration de crr::European< TPayoff >:



Fonctions membres publiques

- **European** (double **S0**, double **R**, double **sigma**, double **T**, int **N**, const **TPayoff &payoff**)
- **std::vector< std::vector< double > > treePrice () const override**

Arbre des prix de l'option selon le modèle CRR.

- `HedgingStrategy hedgingStrategy () const override`

Calcule la couverture optimale (CRR).

Fonctions membres publiques hérités de `crr::Option`

- `Option (double S0, double R, double sigma, double T, int N)`

Constructeur : initialise les paramètres de l'option pour le modèle CRR.

- `double price () const`

Prix initial de l'option.

- `double deltaZero () const`

Delta initial.

Membres hérités additionnels

Attributs protégés hérités de `crr::Option`

- `double S0_`
- `double R_`
- `double sigma_`
- `double T_`

Paramètres initiaux.

- `int N_`

Nombre de pas.

- `double u_`
- `double d_`
- `double discount_`

Paramètres par pas.

6.8.1 Description détaillée

```
template<typename TPayoff>
class crr::European< TPayoff >
```

`Option` européenne CRR.

Paramètres du template

<code>TPayoff</code>	Type de payoff.
----------------------	-----------------

6.8.2 Documentation des constructeurs et destructeur

6.8.2.1 `European()`

```
template<typename TPayoff>
crr::European< TPayoff >::European (
    double s0,
    double R,
    double sigma,
    double T,
    int N,
    const TPayoff & payoff)
```

Voici le graphe d'appel pour cette fonction :



6.8.3 Documentation des fonctions membres

6.8.3.1 `hedgingStrategy()`

```
template<typename TPayoff>
Option::HedgingStrategy crr::European< TPayoff >::hedgingStrategy () const [override], [virtual]
```

Calcule la couverture optimale (CRR).

Renvoie

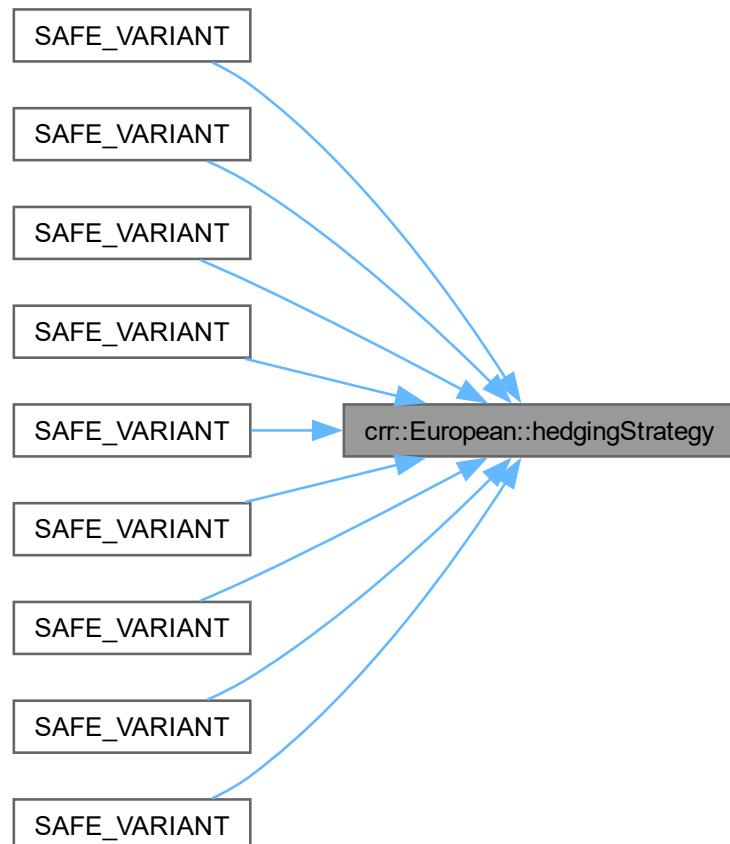
[HedgingStrategy](#) contenant delta et bond par nœud.

Implémente [crr::Option](#).

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.8.3.2 treePrice()

```
template<typename TPayoff>
std::vector< std::vector< double > > crr::European< TPayoff >::treePrice () const [override],
[virtual]
```

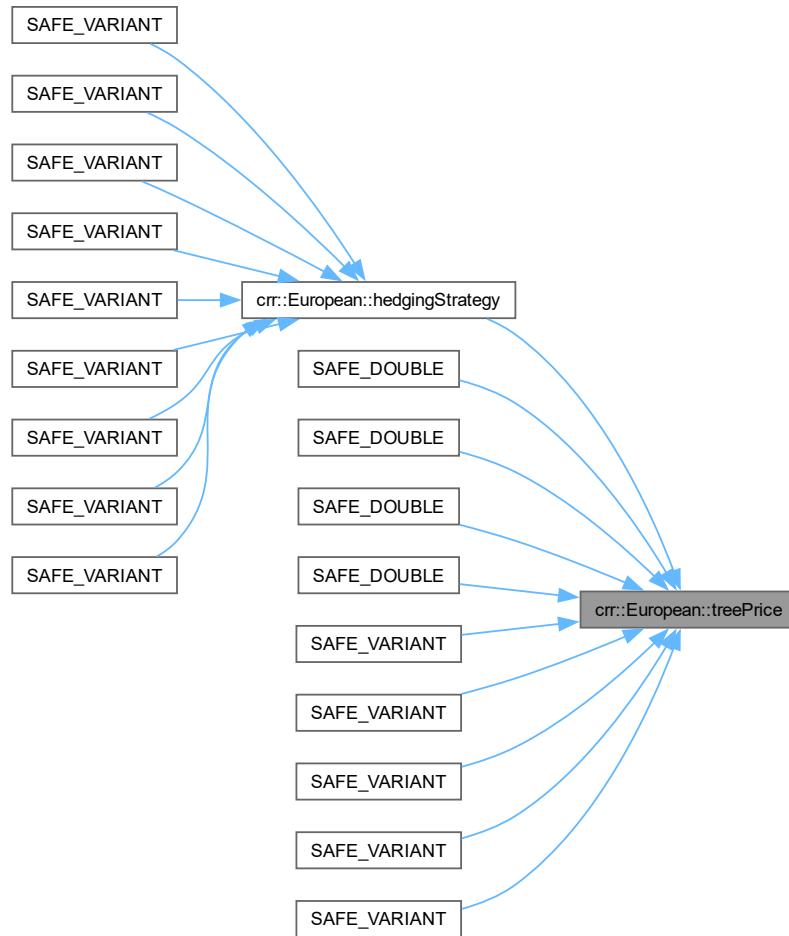
Arbre des prix de l'option selon le modèle CRR.

Renvoie

Matrice des valeurs aux nœuds.

Implémente [crr::Option](#).

Voici le graphe des appelants de cette fonction :



La documentation de cette classe a été générée à partir du fichier suivant :

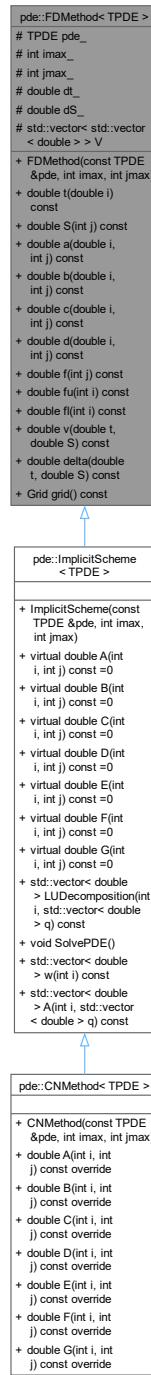
— [src/European.h](#)

6.9 Référence du modèle de la classe pde::FDMETHOD< TPDE >

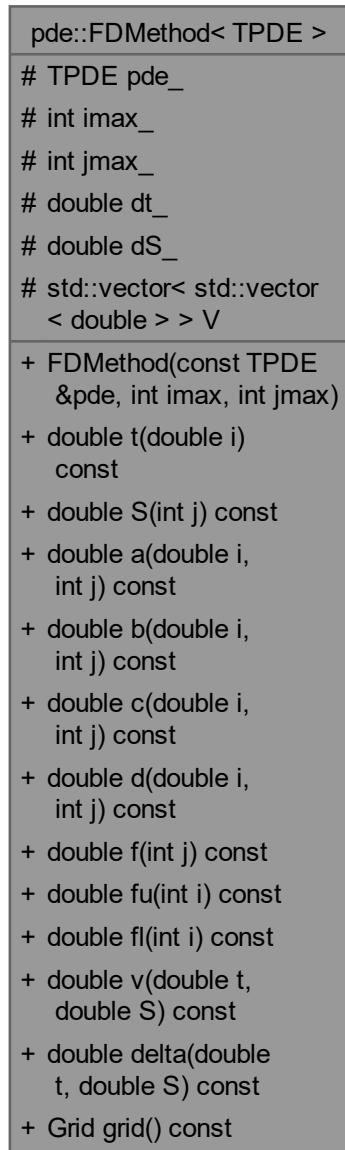
Méthodes aux différences finies pour la résolution numérique d'EDP.

```
#include <FDMETHOD.h>
```

Graphe d'héritage de pde::FDMMethod< TPDE >:



Graphe de collaboration de pde::FDMethod< TPDE >:



Classes

- struct [Grid](#)

Grille : prix et delta.

Fonctions membres publiques

- [FDMethod](#) (const TPDE &pde, int [imax](#), int [jmax](#))

- double **t** (double i) const
- double **S** (int j) const
- double **a** (double i, int j) const
- double **b** (double i, int j) const
- double **c** (double i, int j) const
- double **d** (double i, int j) const
- double **f** (int j) const
- double **fu** (int i) const
- double **fl** (int i) const
- double **v** (double **t**, double **S**) const

Prix de l'option par interpolation bilinéaire.

- double **delta** (double **t**, double **S**) const

Delta de l'option par schéma aux différences centrales.

- **Grid grid** () const

Génère un maillage fixe de 11×11 points pour (t,S).

Attributs protégés

- TPDE **pde_**
- int **imax_**
- int **jmax_**

Nombre de pas en temps et en prix.

- double **dt_**
- double **dS_**

Pas en temps et en prix.

- std::vector< std::vector< double > > **V**

Matrice de solution.

6.9.1 Description détaillée

```
template<typename TPDE>
class pde::FDMethod< TPDE >
```

Méthodes aux différences finies pour la résolution numérique d'EDP.

Paramètres du template

<i>TPDE</i>	Type d'EDP.
-------------	-------------

6.9.2 Documentation des constructeurs et destructeur

6.9.2.1 FDMETHOD()

```
template<typename TPDE>
pde::FDMETHOD< TPDE >::FDMETHOD (
    const TPDE & pde,
    int imax,
    int jmax)
```

Voici le graphe des appels de cette fonction :

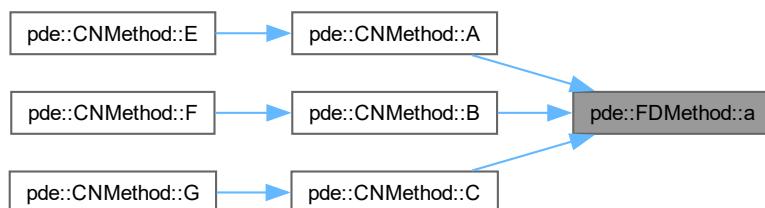


6.9.3 Documentation des fonctions membres

6.9.3.1 a()

```
template<typename TPDE>
double pde::FDMETHOD< TPDE >::a (
    double i,
    int j) const [inline]
```

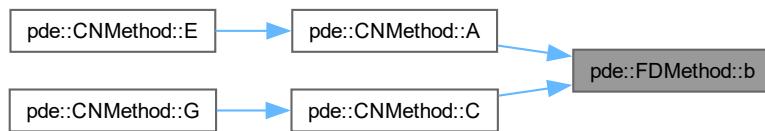
Voici le graphe des appels de cette fonction :



6.9.3.2 b()

```
template<typename TPDE>
double pde::FDMethod< TPDE >::b (
    double i,
    int j) const [inline]
```

Voici le graphe des appelants de cette fonction :



6.9.3.3 c()

```
template<typename TPDE>
double pde::FDMethod< TPDE >::c (
    double i,
    int j) const [inline]
```

Voici le graphe des appelants de cette fonction :



6.9.3.4 d()

```
template<typename TPDE>
double pde::FDMethod< TPDE >::d (
    double i,
    int j) const [inline]
```

Voici le graphe des appelants de cette fonction :



6.9.3.5 delta()

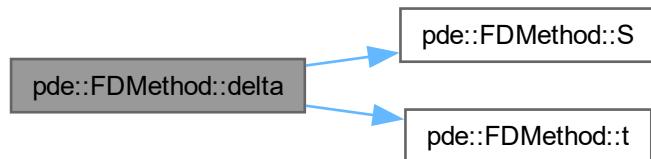
```
template<typename TPDE>
double pde::FDMMethod< TPDE >::delta (
    double t,
    double S) const
```

Delta de l'option par schéma aux différences centrales.

Renvoie

[delta\(t,S\).](#)

Voici le graphe d'appel pour cette fonction :



6.9.3.6 f()

```
template<typename TPDE>
double pde::FDMMethod< TPDE >::f (
    int j) const [inline]
```

Voici le graphe des appels de cette fonction :



6.9.3.7 fl()

```
template<typename TPDE>
double pde::FDMETHOD< TPDE >::fl (
    int i) const [inline]
```

Voici le graphe des appelants de cette fonction :



6.9.3.8 fu()

```
template<typename TPDE>
double pde::FDMETHOD< TPDE >::fu (
    int i) const [inline]
```

Voici le graphe des appelants de cette fonction :



6.9.3.9 grid()

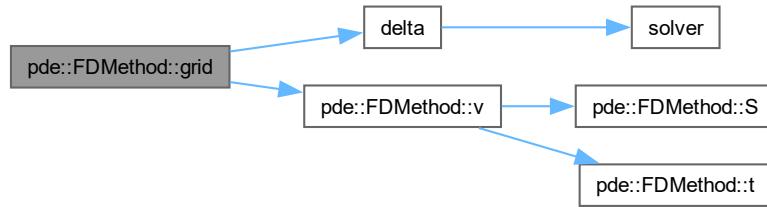
```
template<typename TPDE>
FDMETHOD< TPDE >::Grid pde::FDMETHOD< TPDE >::grid () const
```

Génère un maillage fixe de 11x11 points pour (t,S).

Renvoie

Matrices 11×11 contenant $v(t_i, S_j)$ et $\delta(t_i, S_j)$.

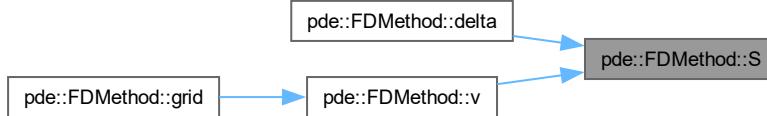
Voici le graphe d'appel pour cette fonction :



6.9.3.10 S()

```
template<typename TPDE>
double pde::FDMethod< TPDE >::S (
    int j) const [inline]
```

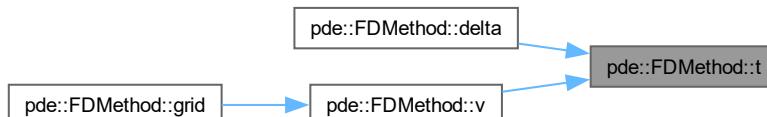
Voici le graphe des appels de cette fonction :



6.9.3.11 t()

```
template<typename TPDE>
double pde::FDMethod< TPDE >::t (
    double i) const [inline]
```

Voici le graphe des appels de cette fonction :



6.9.3.12 v()

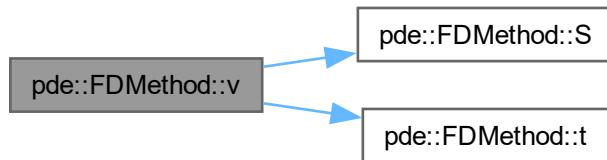
```
template<typename TPDE>
double pde::FDMethod< TPDE >::v (
    double t,
    double S) const
```

Prix de l'option par interpolation bilinéaire.

Renvoie

$v(t, S)$.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appels de cette fonction :



6.9.4 Documentation des données membres

6.9.4.1 dS_

```
template<typename TPDE>
double pde::FDMethod< TPDE >::dS_ [protected]
```

Pas en temps et en prix.

6.9.4.2 dt_

```
template<typename TPDE>
double pde::FDMethod< TPDE >::dt_ [protected]
```

6.9.4.3 imax_

```
template<typename TPDE>
int pde::FDMETHOD< TPDE >::imax_ [protected]
```

6.9.4.4 jmax_

```
template<typename TPDE>
int pde::FDMETHOD< TPDE >::jmax_ [protected]
```

Nombre de pas en temps et en prix.

6.9.4.5 pde_

```
template<typename TPDE>
TPDE pde::FDMETHOD< TPDE >::pde_ [protected]
```

6.9.4.6 V

```
template<typename TPDE>
std::vector<std::vector<double>> pde::FDMETHOD< TPDE >::V [protected]
```

Matrice de solution.

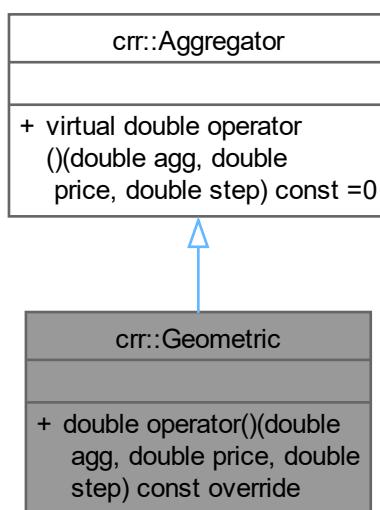
La documentation de cette classe a été générée à partir du fichier suivant :
 — [src/FDMETHOD.h](#)

6.10 Référence de la classe crr::Geometric

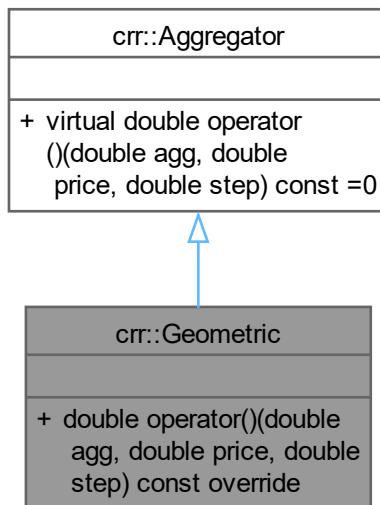
Agrégateur pour la moyenne géométrique.

```
#include <Aggregator.h>
```

Graphe d'héritage de crr::Geometric:



Graphe de collaboration de crr::Geometric:



Fonctions membres publiques

- `double operator()` (double agg, double price, double step) const override

Calcule la nouvelle valeur agrégée.

6.10.1 Description détaillée

Agrégateur pour la moyenne géométrique.

6.10.2 Documentation des fonctions membres

6.10.2.1 `operator()()`

```
double crr::Geometric::operator() (
    double agg,
    double price,
    double step) const [inline], [override], [virtual]
```

Calcule la nouvelle valeur agrégée.

Paramètres

<code>agg</code>	Valeur agrégée jusqu'à l'étape précédente.
<code>price</code>	Prix courant du sous-jacent.
<code>step</code>	Numéro de l'étape.

Renvoie

Valeur agrégée mise à jour.

Implémente [crr::Aggregator](#).

La documentation de cette classe a été générée à partir du fichier suivant :

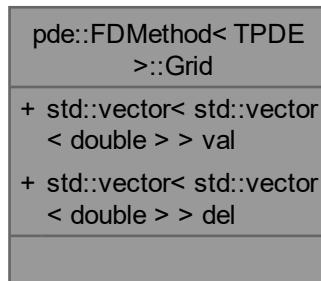
— [src/Aggregator.h](#)

6.11 Référence de la structure pde::FDMethod< TPDE >::Grid

Grille : prix et delta.

```
#include <FDMethod.h>
```

Graphe de collaboration de pde::FDMethod< TPDE >::Grid:



Attributs publics

— `std::vector<std::vector<double>> val`

Prix.

— `std::vector<std::vector<double>> del`

Delta.

6.11.1 Description détaillée

```
template<typename TPDE>
struct pde::FDMethod< TPDE >::Grid
```

Grille : prix et delta.

6.11.2 Documentation des données membres

6.11.2.1 del

```
template<typename TPDE>
std::vector<std::vector<double> > pde::FDMETHOD< TPDE >::Grid::del
```

Delta.

6.11.2.2 val

```
template<typename TPDE>
std::vector<std::vector<double> > pde::FDMETHOD< TPDE >::Grid::val
```

Prix.

La documentation de cette structure a été générée à partir du fichier suivant :

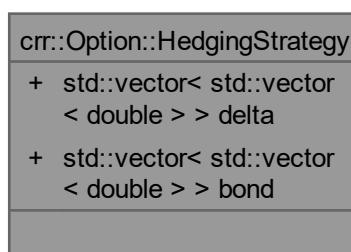
— src/FDMETHOD.h

6.12 Référence de la structure crr::Option::HedgingStrategy

Stratégie de couverture : delta et obligation.

```
#include <Option.h>
```

Graphe de collaboration de crr::Option::HedgingStrategy:



Attributs publics

— std::vector<std::vector<double>> **delta**

Positions en sous-jacent.

— std::vector<std::vector<double>> **bond**

Positions en actif sans risque.

6.12.1 Description détaillée

Stratégie de couverture : delta et obligation.

6.12.2 Documentation des données membres

6.12.2.1 bond

```
std::vector<std::vector<double>> crr::Option::HedgingStrategy::bond
```

Positions en actif sans risque.

6.12.2.2 delta

```
std::vector<std::vector<double>> crr::Option::HedgingStrategy::delta
```

Positions en sous-jacent.

La documentation de cette structure a été générée à partir du fichier suivant :

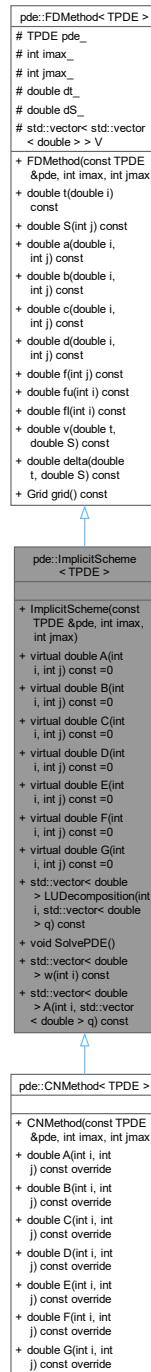
— [src/Option.h](#)

6.13 Référence du modèle de la classe pde::ImplicitScheme< TPDE >

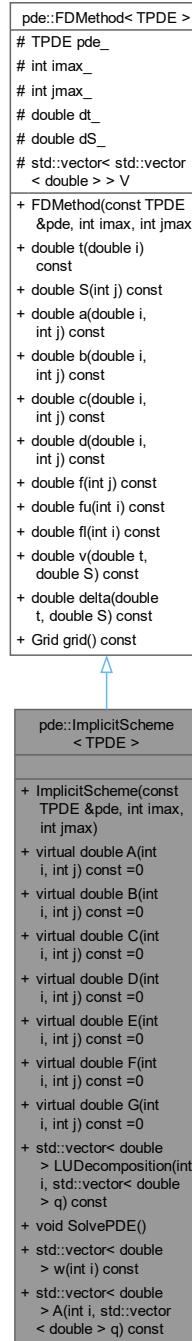
Schéma implicite aux différences finies.

```
#include <ImplicitScheme.h>
```

Graphe d'héritage de pde::ImplicitScheme< TPDE >:



Graphe de collaboration de pde::ImplicitScheme< TPDE >:



Fonctions membres publiques

- [ImplicitScheme](#) (const TPDE &pde, int **imax**, int **jmax**)
- virtual double **A** (int i, int j) const =0

Coefficients de l'équation discrétisée.

- virtual double **B** (int i, int j) const =0
- virtual double **C** (int i, int j) const =0
- virtual double **D** (int i, int j) const =0
- virtual double **E** (int i, int j) const =0
- virtual double **F** (int i, int j) const =0
- virtual double **G** (int i, int j) const =0
- std::vector< double > **LUDecomposition** (int i, std::vector< double > q) const

Résolution d'un système tridiagonal par l'algorithme de Thomas.

- void **SolvePDE** ()

Calcul de la matrice de solution V.

- std::vector< double > **w** (int i) const
- std::vector< double > **A** (int i, std::vector< double > q) const

Fonctions membres publiques hérités de **pde::FDMMethod< TPDE >**

- **FDMMethod** (const TPDE &pde, int **imax**, int **jmax**)
- double **t** (double i) const
- double **S** (int j) const
- double **a** (double i, int j) const
- double **b** (double i, int j) const
- double **c** (double i, int j) const
- double **d** (double i, int j) const
- double **f** (int j) const
- double **fu** (int i) const
- double **fl** (int i) const
- double **v** (double **t**, double **S**) const

Prix de l'option par interpolation bilinéaire.

- double **delta** (double **t**, double **S**) const

Delta de l'option par schéma aux différences centrales.

- **Grid grid** () const

Génère un maillage fixe de 11×11 points pour (t,S).

Membres hérités additionnels

Attributs protégés hérités de pde::FDMETHOD< TPDE >

- TPDE `pde_`
- int `imax_`
- int `jmax_`

Nombre de pas en temps et en prix.

- double `dt_`
- double `dS_`

Pas en temps et en prix.

- std::vector< std::vector< double > > `V`

Matrice de solution.

6.13.1 Description détaillée

```
template<typename TPDE>
class pde::ImplicitScheme< TPDE >
```

Schéma implicite aux différences finies.

Paramètres du template

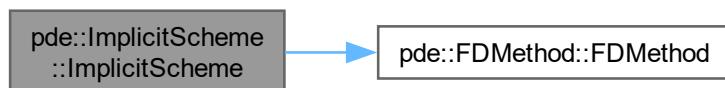
<code>TPDE</code>	Type d'EDP.
-------------------	-------------

6.13.2 Documentation des constructeurs et destructeur

6.13.2.1 ImplicitScheme()

```
template<typename TPDE>
pde::ImplicitScheme< TPDE >::ImplicitScheme (
    const TPDE & pde,
    int imax,
    int jmax)
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.13.3 Documentation des fonctions membres

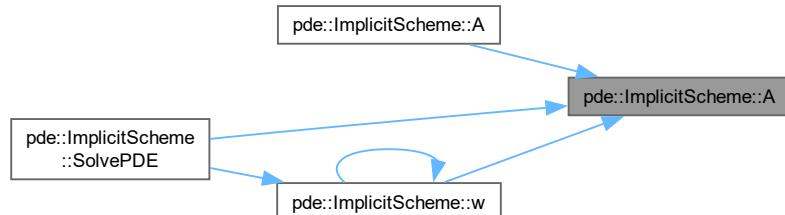
6.13.3.1 A() [1/2]

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::A (
    int i,
    int j) const [pure virtual]
```

Coefficients de l'équation discrétisée.

Implémenté dans `pde::CNMethod< TPDE >`.

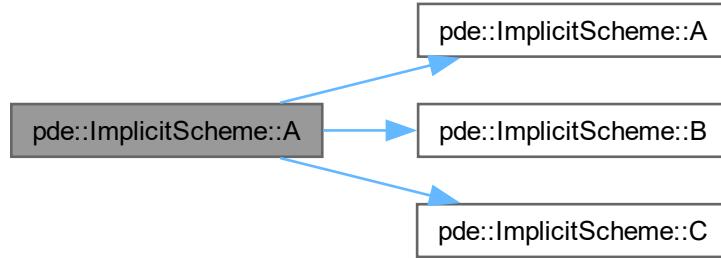
Voici le graphe des appelants de cette fonction :



6.13.3.2 A() [2/2]

```
template<typename TPDE>
std::vector< double > pde::ImplicitScheme< TPDE >::A (
    int i,
    std::vector< double > q) const
```

Voici le graphe d'appel pour cette fonction :



6.13.3.3 B()

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::B (
    int i,
    int j) const [pure virtual]
```

Implémenté dans [pde::CNMethod< TPDE >](#).

Voici le graphe des appels de cette fonction :

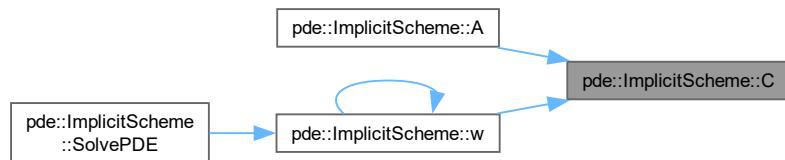


6.13.3.4 C()

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::C (
    int i,
    int j) const [pure virtual]
```

Implémenté dans [pde::CNMethod< TPDE >](#).

Voici le graphe des appelants de cette fonction :



6.13.3.5 D()

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::D (
    int i,
    int j) const [pure virtual]
```

Implémenté dans [pde::CNMethod< TPDE >](#).

Voici le graphe des appelants de cette fonction :

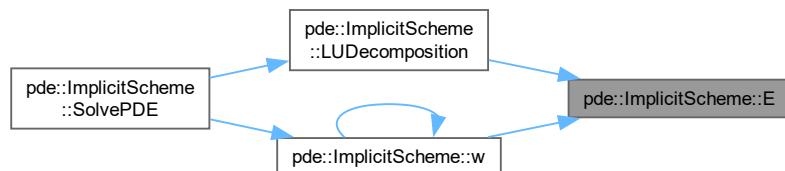


6.13.3.6 E()

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::E (
    int i,
    int j) const [pure virtual]
```

Implémenté dans [pde::CNMethod< TPDE >](#).

Voici le graphe des appelants de cette fonction :

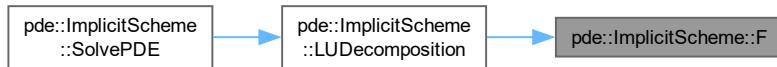


6.13.3.7 F()

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::F (
    int i,
    int j) const [pure virtual]
```

Implémenté dans [pde::CNMethod< TPDE >](#).

Voici le graphe des appelants de cette fonction :



6.13.3.8 G()

```
template<typename TPDE>
virtual double pde::ImplicitScheme< TPDE >::G (
    int i,
    int j) const [pure virtual]
```

Implémenté dans [pde::CNMethod< TPDE >](#).

6.13.3.9 LUDecomposition()

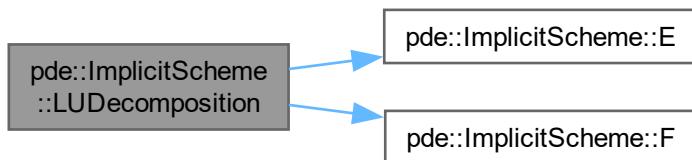
```
template<typename TPDE>
std::vector< double > pde::ImplicitScheme< TPDE >::LUDecomposition (
    int i,
    std::vector< double > q) const
```

Résolution d'un système tridiagonal par l'algorithme de Thomas.

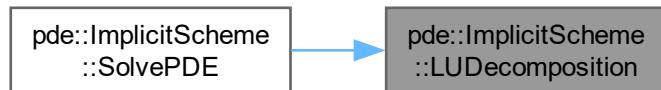
Renvoie

Vecteur solution du système.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

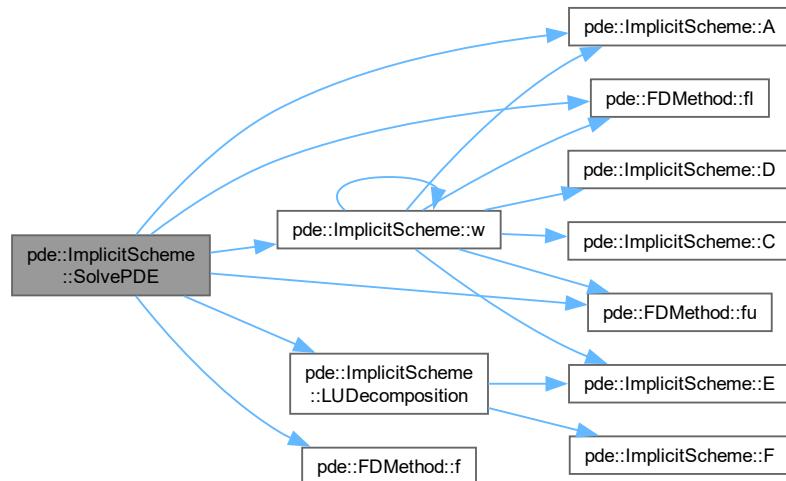


6.13.3.10 SolvePDE()

```
template<typename TPDE>
void pde::ImplicitScheme< TPDE >::SolvePDE ()
```

Calcul de la matrice de solution V.

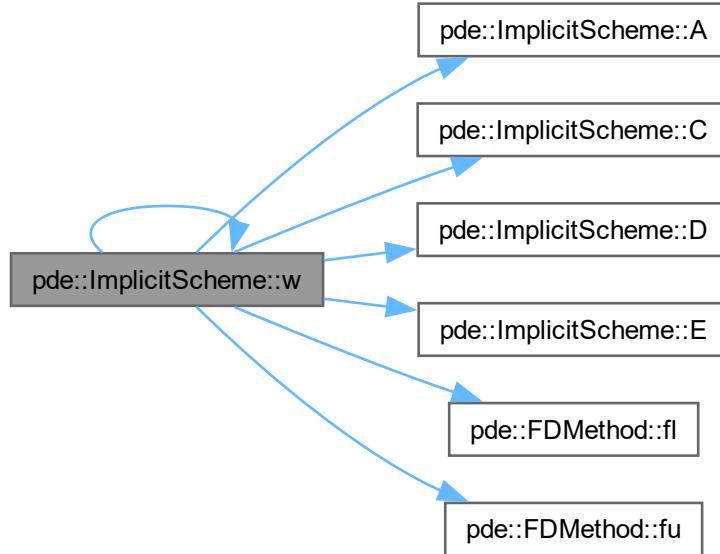
Voici le graphe d'appel pour cette fonction :



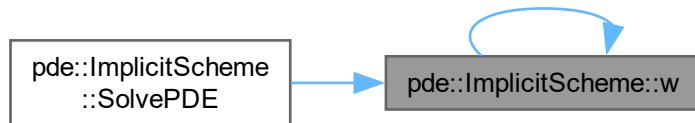
6.13.3.11 w()

```
template<typename TPDE>
std::vector< double > pde::ImplicitScheme< TPDE >::w (
    int i) const
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



La documentation de cette classe a été générée à partir du fichier suivant :

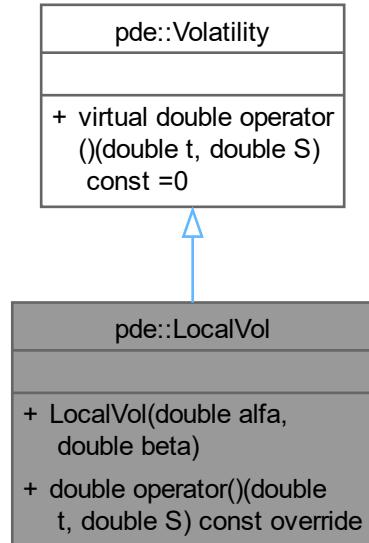
— `src/ImplicitScheme.h`

6.14 Référence de la classe pde::LocalVol

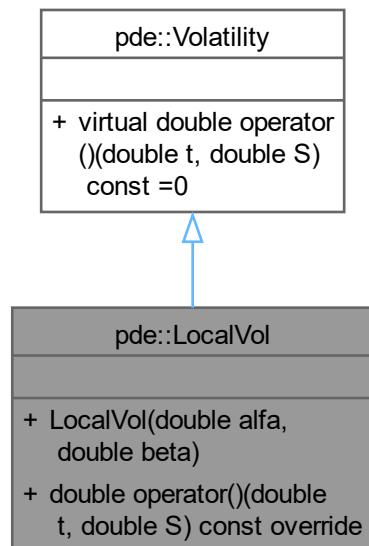
Volatilité dans un modèle à volatilité locale.

```
#include <Volatility.h>
```

Graphe d'héritage de pde::LocalVol:



Graphe de collaboration de pde::LocalVol:



Fonctions membres publiques

- [LocalVol \(double alfa, double beta\)](#)
- [double operator\(\) \(double t, double S\) const override](#)

Calcule la valeur de la volatilité pour un prix S du sous-jacent à l'instant t.

6.14.1 Description détaillée

Volatilité dans un modèle à volatilité locale.

6.14.2 Documentation des constructeurs et destructeur

6.14.2.1 LocalVol()

```
pde::LocalVol::LocalVol (
    double alfa,
    double beta)
```

Paramètres

<i>alfa</i>	Coefficient du temps.
<i>beta</i>	Coefficient du sous-jacent.

6.14.3 Documentation des fonctions membres

6.14.3.1 operator()()

```
double pde::LocalVol::operator() (
    double t,
    double S) const [override], [virtual]
```

Calcule la valeur de la volatilité pour un prix S du sous-jacent à l'instant t.

Paramètres

<i>t</i>	Temps.
<i>S</i>	Prix du sous-jacent.

Renvoie

Valeur de la volatilité.

Implémente [pde::Volatility](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

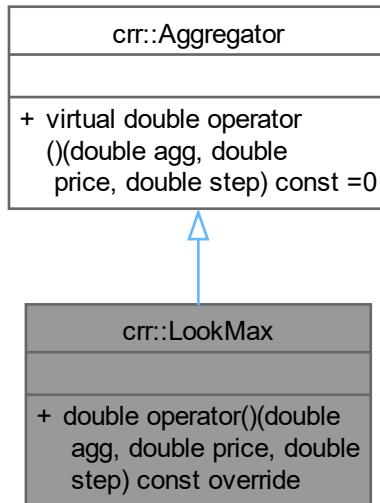
- [src/Volatility.h](#)
- [src/Volatility.cpp](#)

6.15 Référence de la classe crr::LookMax

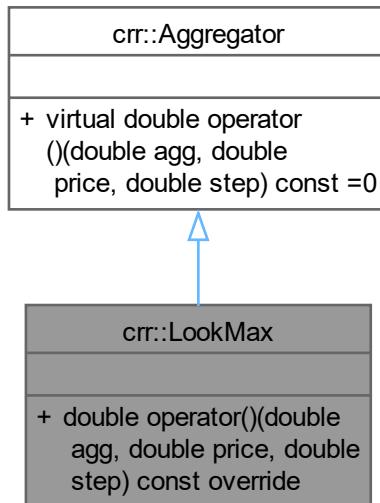
Agrégateur pour le maximum.

```
#include <Aggregator.h>
```

Graphe d'héritage de crr::LookMax:



Graphe de collaboration de `crr::LookMax`:



Fonctions membres publiques

— double **operator()** (double agg, double price, double step) const override

Calcule la nouvelle valeur agrégée.

6.15.1 Description détaillée

Agrégateur pour le maximum.

6.15.2 Documentation des fonctions membres

6.15.2.1 operator()()

```
double crr::LookMax::operator() (  
    double agg,  
    double price,  
    double step) const [inline], [override], [virtual]
```

Calcule la nouvelle valeur agrégée.

Paramètres

<i>agg</i>	Valeur agrégée jusqu'à l'étape précédente.
<i>price</i>	Prix courant du sous-jacent.
<i>step</i>	Numéro de l'étape.

Renvoie

Valeur agrégée mise à jour.

Implémente [crr::Aggregator](#).

La documentation de cette classe a été générée à partir du fichier suivant :

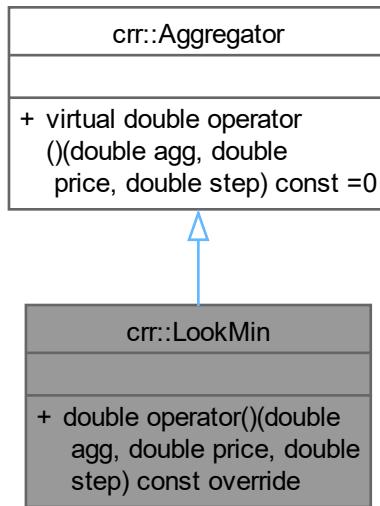
— [src/Aggregator.h](#)

6.16 Référence de la classe crr::LookMin

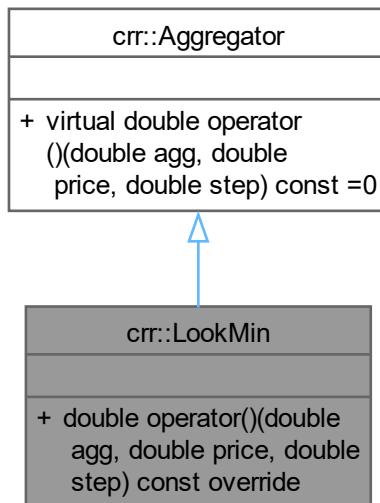
Agrégateur pour le minimum.

```
#include <Aggregator.h>
```

Graphe d'héritage de crr::LookMin:



Graphe de collaboration de crr::LookMin:



Fonctions membres publiques

- double **operator()** (double agg, double price, double step) const override

Calcule la nouvelle valeur agrégée.

6.16.1 Description détaillée

Agrégateur pour le minimum.

6.16.2 Documentation des fonctions membres

6.16.2.1 operator()()

```
double crr::LookMin::operator() (
    double agg,
    double price,
    double step) const [inline], [override], [virtual]
```

Calcule la nouvelle valeur agrégée.

Paramètres

<i>agg</i>	Valeur agrégée jusqu'à l'étape précédente.
<i>price</i>	Prix courant du sous-jacent.
<i>step</i>	Numéro de l'étape.

Renvoie

Valeur agrégée mise à jour.

Implémente [crr::Aggregator](#).

La documentation de cette classe a été générée à partir du fichier suivant :

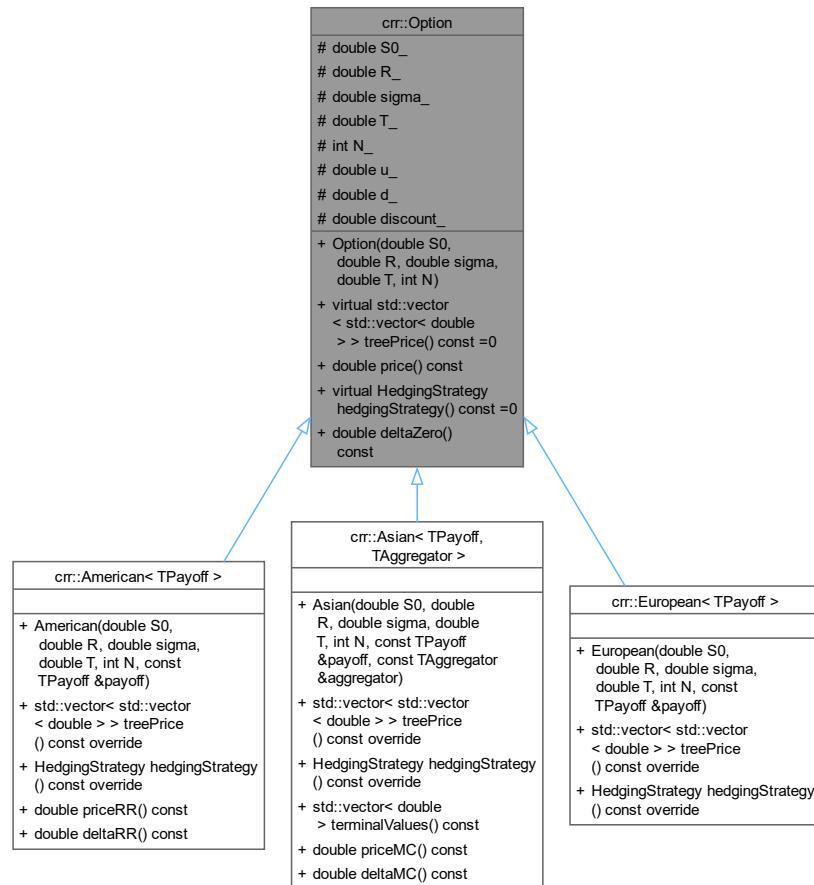
— [src/Aggregator.h](#)

6.17 Référence de la classe crr::Option

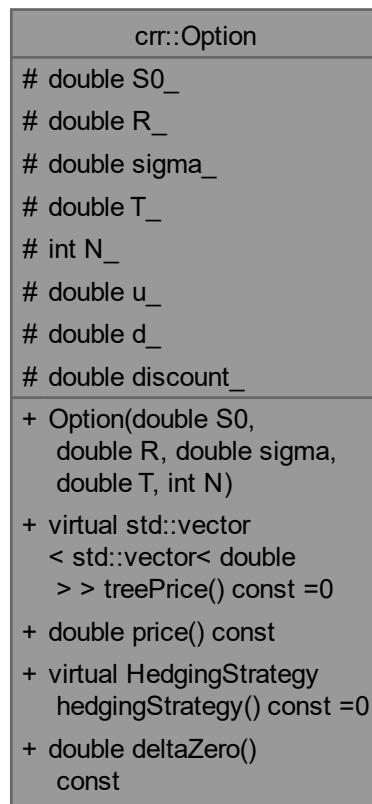
Classe de base [Option](#) : calcul des paramètres du modèle.

```
#include <Option.h>
```

Graphe d'héritage de crr::Option:



Graphe de collaboration de crr::Option:



Classes

- struct [HedgingStrategy](#)

Stratégie de couverture : delta et obligation.

Fonctions membres publiques

- [Option](#) (double **S0**, double **R**, double **sigma**, double **T**, int **N**)

Constructeur : initialise les paramètres de l'option pour le modèle CRR.

- virtual std::vector<std::vector<double>> [treePrice](#) () const =0

Arbre des prix de l'option selon le modèle CRR.

- double [price](#) () const

Prix initial de l'option.

— virtual [HedgingStrategy hedgingStrategy \(\) const =0](#)

Calcule la couverture optimale (CRR).

— double [deltaZero \(\) const](#)

Delta initial.

Attributs protégés

— double [S0_](#)

— double [R_](#)

— double [sigma_](#)

— double [T_](#)

Paramètres initiaux.

— int [N_](#)

Nombre de pas.

— double [u_](#)

— double [d_](#)

— double [discount_](#)

Paramètres par pas.

6.17.1 Description détaillée

Classe de base [Option](#) : calcul des paramètres du modèle.

6.17.2 Documentation des constructeurs et destructeur

6.17.2.1 Option()

```
crr::Option::Option (
    double S0,
    double R,
    double sigma,
    double T,
    int N)
```

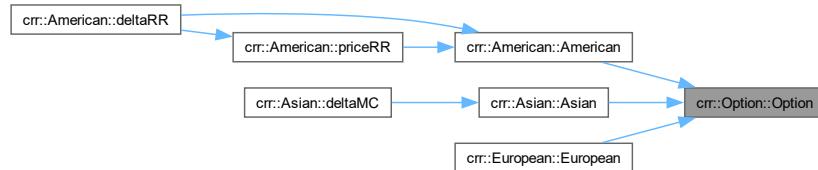
Constructeur : initialise les paramètres de l'option pour le modèle CRR.

Paramètres

<i>S0</i>	Prix initial du sous-jacent.
<i>R</i>	Taux d'intérêt sans risque continu.

<i>sigma</i>	Volatilité annuelle du sous-jacent.
<i>T</i>	Durée jusqu'à l'échéance en années.
<i>N</i>	Nombre de pas de l'arbre binomial.

Voici le graphe des appels de cette fonction :



6.17.3 Documentation des fonctions membres

6.17.3.1 deltaZero()

```
double crr::Option::deltaZero () const
```

Delta initial.

Renvoie

Valeur du delta à n = 0.

Voici le graphe d'appel pour cette fonction :



6.17.3.2 hedgingStrategy()

```
virtual HedgingStrategy crr::Option::hedgingStrategy () const [pure virtual]
```

Calcule la couverture optimale (CRR).

Renvoie

[HedgingStrategy](#) contenant delta et bond par nœud.

Implémenté dans [crr::American< TPayoff >](#), [crr::Asian< TPayoff, TAggregator >](#), et [crr::European< TPayoff >](#).

Voici le graphe des appelants de cette fonction :

**6.17.3.3 price()**

```
double crr::Option::price () const
```

Prix initial de l'option.

Renvoie

Valeur de l'option à $n = 0$.

Voici le graphe d'appel pour cette fonction :

**6.17.3.4 treePrice()**

```
virtual std::vector< std::vector< double > > crr::Option::treePrice () const [pure virtual]
```

Arbre des prix de l'option selon le modèle CRR.

Renvoie

Matrice des valeurs aux nœuds.

Implémenté dans [crr::American< TPayoff >](#), [crr::Asian< TPayoff, TAggregator >](#), et [crr::European< TPayoff >](#).

Voici le graphe des appelants de cette fonction :



6.17.4 Documentation des données membres

6.17.4.1 d_

```
double crr::Option::d_ [protected]
```

6.17.4.2 discount_

```
double crr::Option::discount_ [protected]
```

Paramètres par pas.

6.17.4.3 N_

```
int crr::Option::N_ [protected]
```

Nombre de pas.

6.17.4.4 R_

```
double crr::Option::R_ [protected]
```

6.17.4.5 S0_

```
double crr::Option::S0_ [protected]
```

6.17.4.6 sigma_

```
double crr::Option::sigma_ [protected]
```

6.17.4.7 T_

```
double crr::Option::T_ [protected]
```

Paramètres initiaux.

6.17.4.8 u_

```
double crr::Option::u_ [protected]
```

La documentation de cette classe a été générée à partir des fichiers suivants :

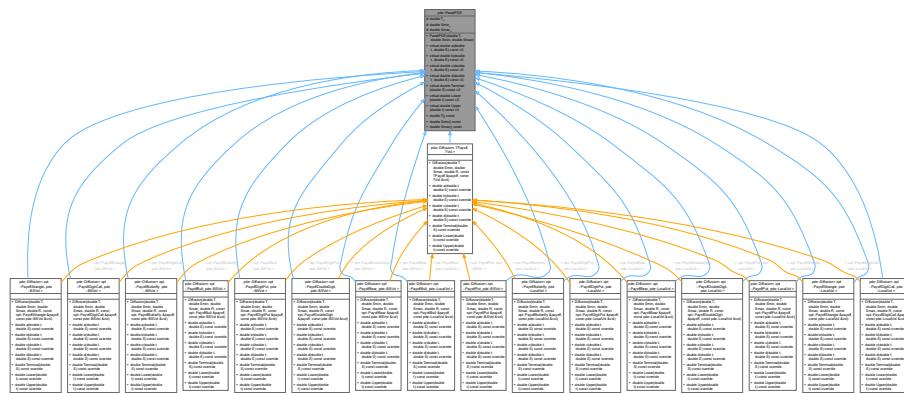
- src/[Option.h](#)
- src/[Option.cpp](#)

6.18 Référence de la classe pde::ParabPDE

Interface pour EDP paraboliques de type général.

```
#include <ParabPDE.h>
```

Graphe d'héritage de pde::ParabPDE:



Graphe de collaboration de pde::ParabPDE:

pde::ParabPDE	
# double	T_
# double	Smin_
# double	Smax_
+ ParabPDE	(double T, double Smin, double Smax)
+ virtual double	a(double t, double S) const =0
+ virtual double	b(double t, double S) const =0
+ virtual double	c(double t, double S) const =0
+ virtual double	d(double t, double S) const =0
+ virtual double	Terminal (double S) const =0
+ virtual double	Lower (double t) const =0
+ virtual double	Upper (double t) const =0
+ double	T() const
+ double	Smin() const
+ double	Smax() const

Fonctions membres publiques

- [ParabPDE](#) (double **T**, double **Smin**, double **Smax**)
- virtual double [a](#) (double **t**, double **S**) const =0

Coefficients de l'équation différentielle partielle.

- virtual double [b](#) (double **t**, double **S**) const =0
- virtual double [c](#) (double **t**, double **S**) const =0
- virtual double [d](#) (double **t**, double **S**) const =0
- virtual double [Terminal](#) (double **S**) const =0

Terminal Boundary Condition.

- virtual double [Lower](#) (double **t**) const =0

Lower Boundary Condition.

- virtual double **Upper** (double **t**) const =0

Upper Boundary Condition.

- double **T** () const
- double **Smin** () const
- double **Smax** () const

Attributs protégés

- double **T_**
- double **Smin_**
- double **Smax_**

Domaine de l'EDP.

6.18.1 Description détaillée

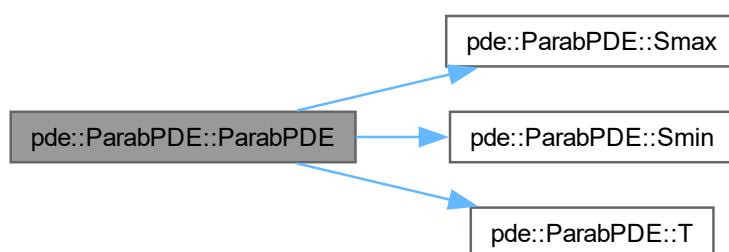
Interface pour EDP paraboliques de type général.

6.18.2 Documentation des constructeurs et destructeur

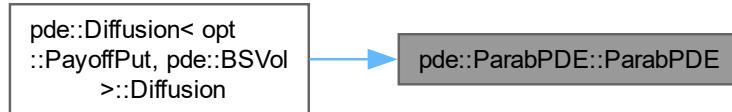
6.18.2.1 ParabPDE()

```
pde::ParabPDE::ParabPDE (
    double T,
    double Smin,
    double Smax) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



6.18.3 Documentation des fonctions membres

6.18.3.1 a()

```
virtual double pde::ParabPDE::a (
    double t,
    double S) const [pure virtual]
```

Coefficients de l'équation différentielle partielle.

Implémenté dans [pde::Diffusion< TPayoff, TVol >](#), [pde::Diffusion< opt::PayoffBear, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBull, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBull, pde::BSVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffStrangle, pde::BSVol >](#), et [pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >](#).

6.18.3.2 b()

```
virtual double pde::ParabPDE::b (
    double t,
    double S) const [pure virtual]
```

Implémenté dans [pde::Diffusion< TPayoff, TVol >](#), [pde::Diffusion< opt::PayoffBear, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBull, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBull, pde::BSVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffStrangle, pde::BSVol >](#), et [pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >](#).

6.18.3.3 c()

```
virtual double pde::ParabPDE::c (
    double t,
    double S) const [pure virtual]
```

Implémenté dans [pde::Diffusion< TPayoff, TVol >](#), [pde::Diffusion< opt::PayoffBear, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBull, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBull, pde::BSVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffStrangle, pde::BSVol >](#), et [pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >](#).

6.18.3.4 d()

```
virtual double pde::ParabPDE::d (
    double t,
    double S) const [pure virtual]
```

Implémenté dans [pde::Diffusion< TPayoff, TVol >, pde::Diffusion< opt::PayoffBear, pde::BSVol >, pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBull, pde::BSVol >, pde::Diffusion< opt::PayoffBull, pde::LocalVol >, pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >, pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >, pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >, pde::Diffusion< opt::PayoffDigitPut, pde::LocalVol >, pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >, pde::Diffusion< opt::PayoffPut, pde::BSVol >, pde::Diffusion< opt::PayoffPut, pde::Diffusion< opt::PayoffStrangle, pde::BSVol >, et pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >](#).

6.18.3.5 Lower()

```
virtual double pde::ParabPDE::Lower (
    double t) const [pure virtual]
```

Lower Boundary Condition.

Renvoie

$v(t, S_{min})$.

Implémenté dans [pde::Diffusion< TPayoff, TVol >, pde::Diffusion< opt::PayoffBear, pde::BSVol >, pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBull, pde::BSVol >, pde::Diffusion< opt::PayoffBull, pde::LocalVol >, pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >, pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >, pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >, pde::Diffusion< opt::PayoffDigitPut, pde::LocalVol >, pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >, pde::Diffusion< opt::PayoffPut, pde::BSVol >, pde::Diffusion< opt::PayoffPut, pde::Diffusion< opt::PayoffStrangle, pde::BSVol >, et pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >](#).

6.18.3.6 Smax()

```
double pde::ParabPDE::Smax () const [inline]
```

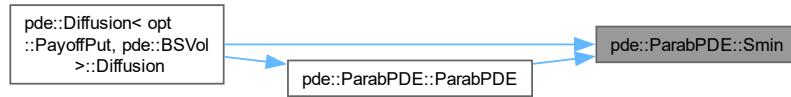
Voici le graphe des appels de cette fonction :



6.18.3.7 Smin()

```
double pde::ParabPDE::Smin () const [inline]
```

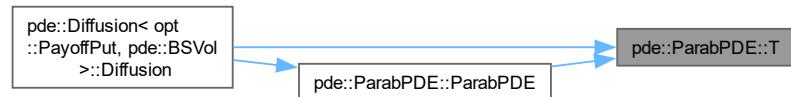
Voici le graphe des appelants de cette fonction :



6.18.3.8 T()

```
double pde::ParabPDE::T () const [inline]
```

Voici le graphe des appelants de cette fonction :



6.18.3.9 Terminal()

```
virtual double pde::ParabPDE::Terminal (
    double S) const [pure virtual]
```

Terminal Boundary Condition.

Renvoie

$v(T, S)$.

Implémenté dans [pde::Diffusion< TPayoff, TVol >](#), [pde::Diffusion< opt::PayoffBear, pde::BSVol >](#), [pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBear, pde::BSVol > >](#), [pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBear, pde::BSVol > >](#), [pde::Diffusion< opt::PayoffBear, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::BSVol > >](#), [pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >](#), [pde::Diffusion< opt::PayoffPut, pde::BSVol >](#), [pde::Diffusion< opt::PayoffStrangle, pde::BSVol >](#), et [pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >](#).

6.18.3.10 Upper()

```
virtual double pde::ParabPDE::Upper (
    double t) const [pure virtual]
```

Upper Boundary Condition.

Renvoie

$v(t, S_{max})$.

Implémenté dans `pde::Diffusion< TPayoff, TVol >`, `pde::Diffusion< opt::PayoffBear, pde::BSVol >`, `pde::Diffusion< opt::PayoffBear, pde::Diffusion< opt::PayoffBull, pde::BSVol >`, `pde::Diffusion< opt::PayoffBull, pde::LocalVol >`, `pde::Diffusion< opt::PayoffButterfly, pde::Diffusion< opt::PayoffButterfly, pde::LocalVol >`, `pde::Diffusion< opt::PayoffDigitCall, pde::BSVol >`, `pde::Diffusion< opt::PayoffDigitPut, pde::BSVol >`, `pde::Diffusion< opt::PayoffDigitPut, pde::LocalVol >`, `pde::Diffusion< opt::PayoffDoubleDigit, pde::LocalVol >`, `pde::Diffusion< opt::PayoffPut, pde::BSVol >`, `pde::Diffusion< opt::PayoffPut, pde::Diffusion< opt::PayoffStrangle, pde::BSVol >`, et `pde::Diffusion< opt::PayoffStrangle, pde::LocalVol >`.

6.18.4 Documentation des données membres

6.18.4.1 Smax_

```
double pde::ParabPDE::Smax_ [protected]
```

Domaine de l'EDP.

6.18.4.2 Smin_

```
double pde::ParabPDE::Smin_ [protected]
```

6.18.4.3 T_

```
double pde::ParabPDE::T_ [protected]
```

La documentation de cette classe a été générée à partir du fichier suivant :

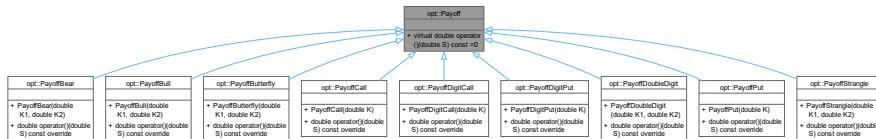
— [src/ParabPDE.h](#)

6.19 Référence de la classe opt::Payoff

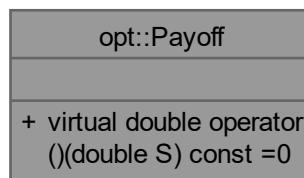
Classe abstraite représentant le payoff d'une option.

```
#include <Payoff.h>
```

Graphe d'héritage de opt::Payoff:



Graphe de collaboration de opt::Payoff:



Fonctions membres publiques

- virtual double **operator()** (double **S**) const =0

Calcule la valeur du payoff pour un prix S du sous-jacent.

6.19.1 Description détaillée

Classe abstraite représentant le payoff d'une option.

6.19.2 Documentation des fonctions membres

6.19.2.1 operator()()

```
virtual double opt::Payoff::operator() (
    double S) const [pure virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémenté dans [opt::PayoffBear](#), [opt::PayoffBull](#), [opt::PayoffButterfly](#), [opt::PayoffCall](#), [opt::PayoffDigitCall](#), [opt::PayoffDigitPut](#), [opt::PayoffDoubleDigit](#), [opt::PayoffPut](#), et [opt::PayoffStrangle](#).

La documentation de cette classe a été générée à partir du fichier suivant :

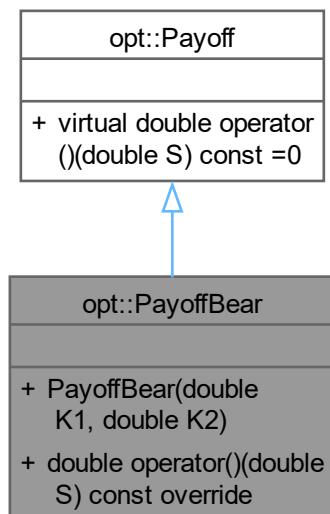
— [src/Payoff.h](#)

6.20 Référence de la classe opt::PayoffBear

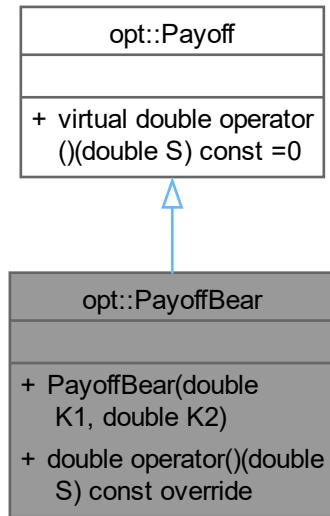
[Payoff](#) d'une option bear spread.

```
#include <Payoff.h>
```

Graphe d'héritage de opt::PayoffBear:



Graphe de collaboration de opt::PayoffBear:



Fonctions membres publiques

- `PayoffBear` (double **K1**, double **K2**)
- `double operator()` (double **S**) const override

Calcule la valeur du payoff pour un prix S du sous-jacent.

6.20.1 Description détaillée

`Payoff` d'une option bear spread.

6.20.2 Documentation des constructeurs et destructeur

6.20.2.1 `PayoffBear()`

```
opt::PayoffBear::PayoffBear (
    double K1,
    double K2)
```

6.20.3 Documentation des fonctions membres

6.20.3.1 `operator()()`

```
double opt::PayoffBear::operator() (
    double S) const [override], [virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

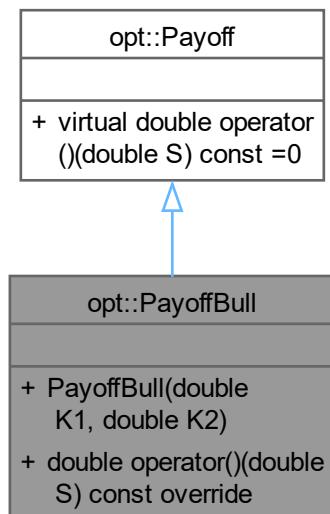
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.21 Référence de la classe opt::PayoffBull

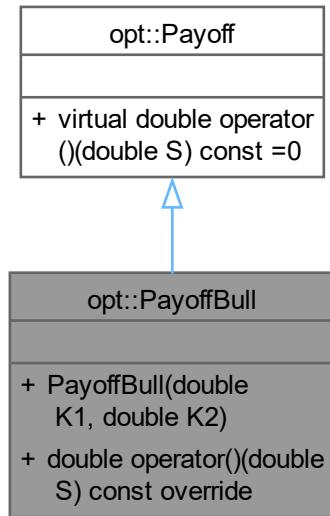
[Payoff](#) d'une option bull spread.

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffBull](#):



Graphe de collaboration de opt::PayoffBull:



Fonctions membres publiques

- [PayoffBull](#) (double **K1**, double **K2**)
- double [operator\(\)](#) (double **S**) const override

*Calcule la valeur du payoff pour un prix **S** du sous-jacent.*

6.21.1 Description détaillée

[Payoff](#) d'une option bull spread.

6.21.2 Documentation des constructeurs et destructeur

6.21.2.1 [PayoffBull\(\)](#)

```
opt::PayoffBull::PayoffBull (
    double K1,
    double K2)
```

6.21.3 Documentation des fonctions membres

6.21.3.1 [operator\(\)](#)

```
double opt::PayoffBull::operator() (
    double S) const [override], [virtual]
```

*Calcule la valeur du payoff pour un prix **S** du sous-jacent.*

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

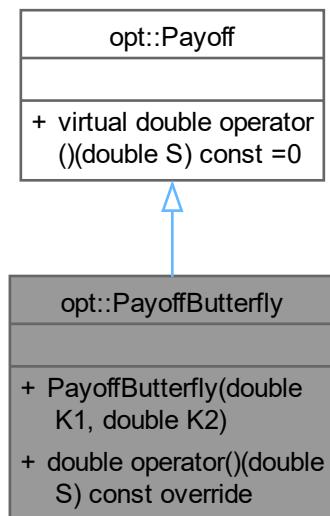
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.22 Référence de la classe opt::PayoffButterfly

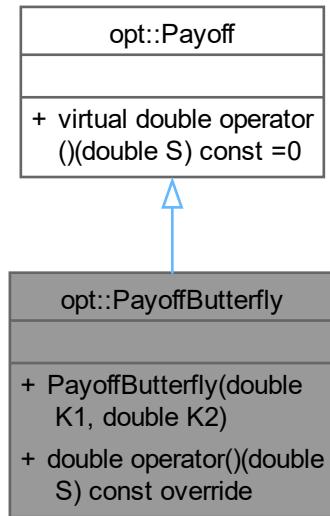
[Payoff](#) d'une option butterfly.

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffButterfly](#):



Graphe de collaboration de opt::PayoffButterfly:



Fonctions membres publiques

- `PayoffButterfly` (double **K1**, double **K2**)
- `double operator()` (double **S**) const override

*Calcule la valeur du payoff pour un prix **S** du sous-jacent.*

6.22.1 Description détaillée

[Payoff](#) d'une option butterfly.

6.22.2 Documentation des constructeurs et destructeur

6.22.2.1 `PayoffButterfly()`

```
opt::PayoffButterfly::PayoffButterfly (
    double K1,
    double K2)
```

6.22.3 Documentation des fonctions membres

6.22.3.1 `operator()`

```
double opt::PayoffButterfly::operator() (
    double S) const [override], [virtual]
```

*Calcule la valeur du payoff pour un prix **S** du sous-jacent.*

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

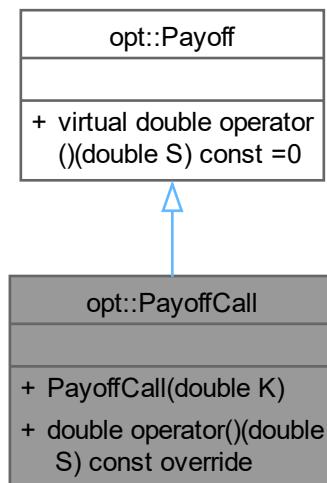
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.23 Référence de la classe opt::PayoffCall

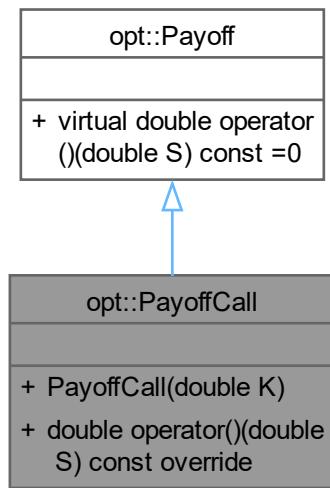
[Payoff](#) d'une option d'achat européenne (call).

```
#include <Payoff.h>
```

Graphe d'héritage de opt::PayoffCall:



Graphe de collaboration de opt::PayoffCall:



Fonctions membres publiques

- [PayoffCall](#) (double *K*)
- double [operator\(\)](#) (double *S*) const override

*Calcule la valeur du payoff pour un prix *S* du sous-jacent.*

6.23.1 Description détaillée

[Payoff](#) d'une option d'achat européenne (call).

6.23.2 Documentation des constructeurs et destructeur

6.23.2.1 PayoffCall()

```
opt::PayoffCall::PayoffCall (
    double K)
```

Paramètres

<i>K</i>	Prix d'exercice de l'option.
----------	------------------------------

6.23.3 Documentation des fonctions membres

6.23.3.1 operator()()

```
double opt::PayoffCall::operator() (
    double S) const [override], [virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

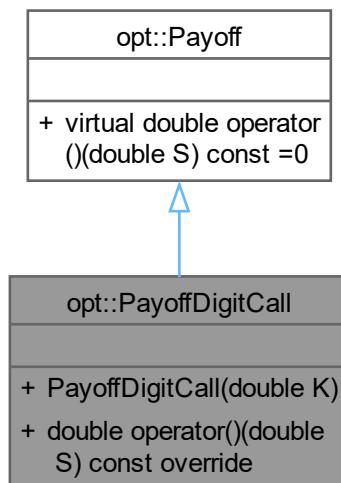
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.24 Référence de la classe opt::PayoffDigitCall

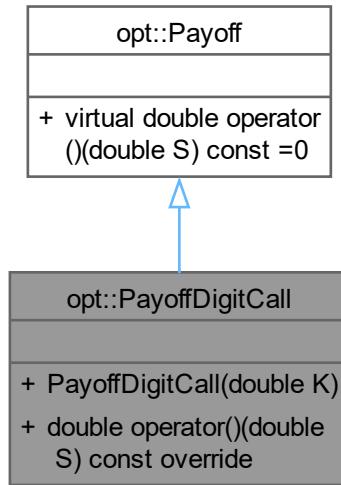
[Payoff](#) d'un call digital.

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffDigitCall](#):



Graphe de collaboration de opt::PayoffDigitCall:



Fonctions membres publiques

- `PayoffDigitCall (double K)`
- `double operator() (double S) const override`

Calcule la valeur du payoff pour un prix S du sous-jacent.

6.24.1 Description détaillée

[Payoff](#) d'un call digital.

6.24.2 Documentation des constructeurs et destructeur

6.24.2.1 `PayoffDigitCall()`

```
opt::PayoffDigitCall::PayoffDigitCall (
    double K)
```

6.24.3 Documentation des fonctions membres

6.24.3.1 `operator()()`

```
double opt::PayoffDigitCall::operator() (
    double S) const [override], [virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

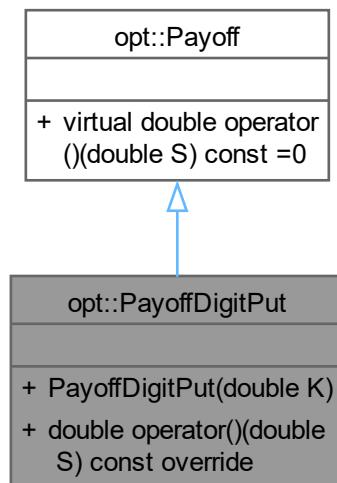
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.25 Référence de la classe opt::PayoffDigitPut

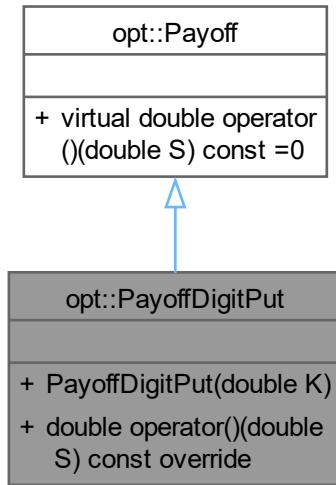
[Payoff](#) d'un put digital.

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffDigitPut](#):



Graphe de collaboration de opt::PayoffDigitPut:



Fonctions membres publiques

- `PayoffDigitPut (double K)`
- `double operator() (double S) const override`

Calcule la valeur du payoff pour un prix S du sous-jacent.

6.25.1 Description détaillée

[Payoff](#) d'un put digital.

6.25.2 Documentation des constructeurs et destructeur

6.25.2.1 `PayoffDigitPut()`

```
opt::PayoffDigitPut::PayoffDigitPut (
    double K)
```

6.25.3 Documentation des fonctions membres

6.25.3.1 `operator()()`

```
double opt::PayoffDigitPut::operator() (
    double S) const [override], [virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

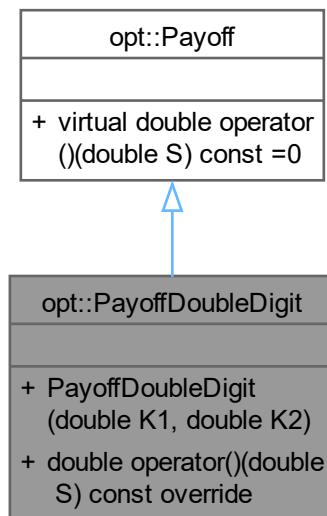
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.26 Référence de la classe opt::PayoffDoubleDigit

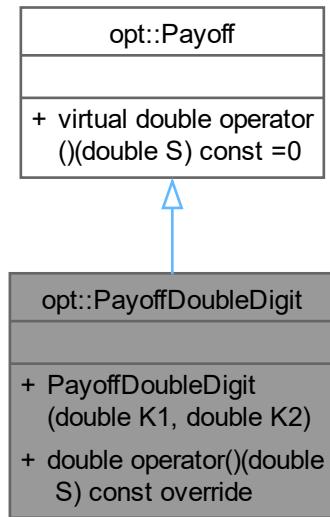
[Payoff](#) d'une option double-digital.

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffDoubleDigit](#):



Graphe de collaboration de opt::PayoffDoubleDigit:



Fonctions membres publiques

- `PayoffDoubleDigit (double K1, double K2)`
- `double operator() (double S) const override`

Calcule la valeur du payoff pour un prix S du sous-jacent.

6.26.1 Description détaillée

Payoff d'une option double-digital.

6.26.2 Documentation des constructeurs et destructeur

6.26.2.1 PayoffDoubleDigit()

```
opt::PayoffDoubleDigit::PayoffDoubleDigit (
    double K1,
    double K2)
```

Paramètres

<code>K1</code>	Prix d'exercice inférieur.
<code>K2</code>	Prix d'exercice supérieur.

6.26.3 Documentation des fonctions membres

6.26.3.1 operator()()

```
double opt::PayoffDoubleDigit::operator() (
    double S) const [override], [virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

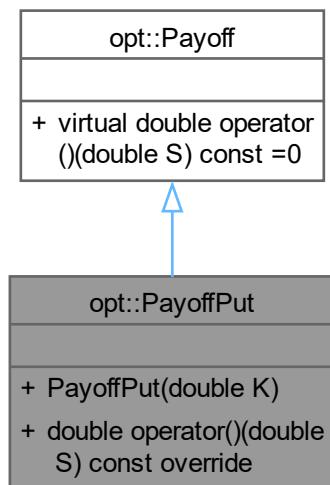
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.27 Référence de la classe opt::PayoffPut

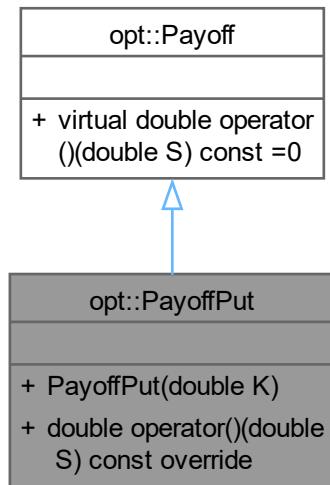
[Payoff](#) d'une option de vente européenne (put).

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffPut](#):



Graphe de collaboration de opt::PayoffPut:



Fonctions membres publiques

- [PayoffPut](#) (double *K*)
- double [operator\(\)](#) (double *S*) const override

*Calcule la valeur du payoff pour un prix *S* du sous-jacent.*

6.27.1 Description détaillée

[Payoff](#) d'une option de vente européenne (put).

6.27.2 Documentation des constructeurs et destructeur

6.27.2.1 PayoffPut()

```
opt::PayoffPut::PayoffPut (
    double K)
```

6.27.3 Documentation des fonctions membres

6.27.3.1 operator()

```
double opt::PayoffPut::operator() (
    double S) const [override], [virtual]
```

*Calcule la valeur du payoff pour un prix *S* du sous-jacent.*

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

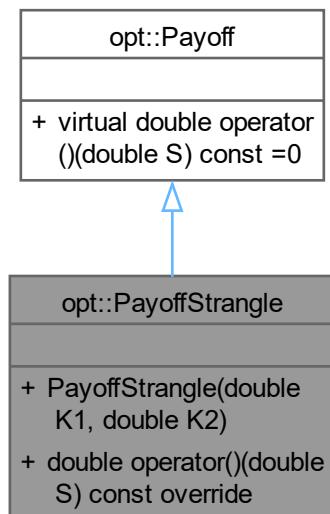
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.28 Référence de la classe opt::PayoffStrangle

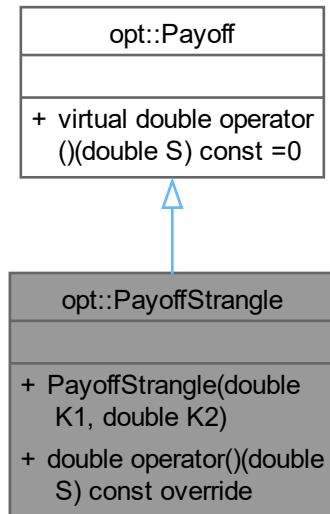
[Payoff](#) d'une option strangle.

```
#include <Payoff.h>
```

Graphe d'héritage de [opt::PayoffStrangle](#):



Graphe de collaboration de opt::PayoffStrangle:



Fonctions membres publiques

- [PayoffStrangle](#) (double **K1**, double **K2**)
- double [operator\(\)](#) (double **S**) const override

Calcule la valeur du payoff pour un prix S du sous-jacent.

6.28.1 Description détaillée

[Payoff](#) d'une option strangle.

6.28.2 Documentation des constructeurs et destructeur

6.28.2.1 [PayoffStrangle\(\)](#)

```
opt::PayoffStrangle::PayoffStrangle (
    double K1,
    double K2)
```

6.28.3 Documentation des fonctions membres

6.28.3.1 [operator\(\)](#)

```
double opt::PayoffStrangle::operator() (
    double S) const [override], [virtual]
```

Calcule la valeur du payoff pour un prix S du sous-jacent.

Paramètres

<i>S</i>	Prix du sous-jacent à maturité.
----------	---------------------------------

Renvoie

Valeur du payoff.

Implémente [opt::Payoff](#).

La documentation de cette classe a été générée à partir des fichiers suivants :

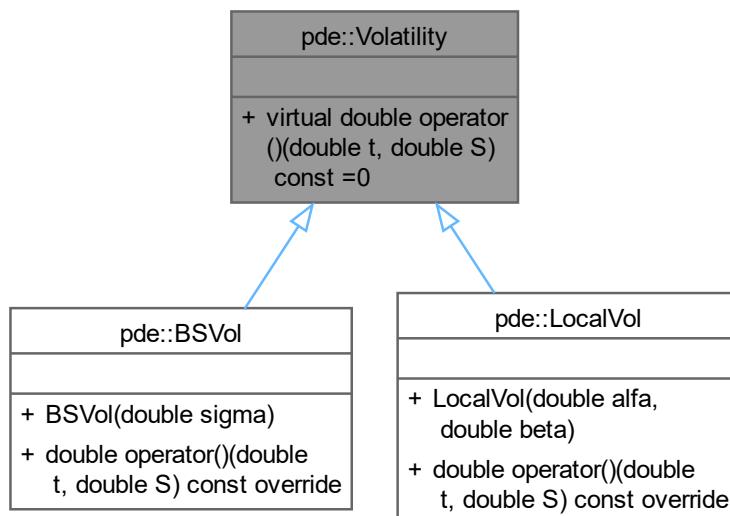
- [src/Payoff.h](#)
- [src/Payoff.cpp](#)

6.29 Référence de la classe pde::Volatility

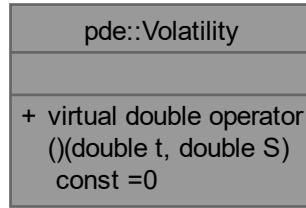
Classe abstraite représentant la volatilité du sous-jacent.

```
#include <Volatility.h>
```

Graphe d'héritage de pde::Volatility:



Graphe de collaboration de pde::Volatility:



Fonctions membres publiques

- virtual double [operator\(\)](#) (double **t**, double **S**) const =0

Calcule la valeur de la volatilité pour un prix S du sous-jacent à l'instant t.

6.29.1 Description détaillée

Classe abstraite représentant la volatilité du sous-jacent.

6.29.2 Documentation des fonctions membres

6.29.2.1 [operator\(\)\(\)](#)

```

virtual double pde::Volatility::operator()
    (double t,
     double S) const [pure virtual]
  
```

Calcule la valeur de la volatilité pour un prix S du sous-jacent à l'instant t.

Paramètres

<i>t</i>	Temps.
<i>S</i>	Prix du sous-jacent.

Renvoie

Valeur de la volatilité.

Implémenté dans [pde::BSVol](#), et [pde::LocalVol](#).

La documentation de cette classe a été générée à partir du fichier suivant :

- [src/Volatility.h](#)

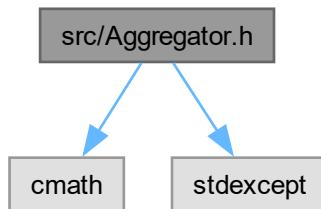
Chapitre 7

Documentation des fichiers

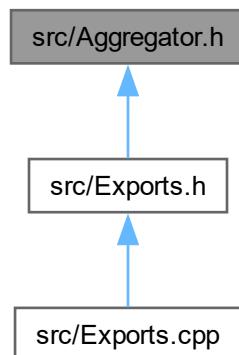
7.1 Référence du fichier src/Aggregator.h

```
#include <cmath>
#include <stdexcept>
```

Graphe des dépendances par inclusion de Aggregator.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

- class [crr::Aggregator](#)

Interface abstraite pour l'agrégation des prix path-dépendants.

- class [crr::Arithmetic](#)

Agrégateur pour la moyenne arithmétique.

- class [crr::Geometric](#)

Agrégateur pour la moyenne géométrique.

- class [crr::LookMax](#)

Agrégateur pour le maximum.

- class [crr::LookMin](#)

Agrégateur pour le minimum.

Espaces de nommage

- namespace [crr](#)

7.2 Aggregator.h

[Aller à la documentation de ce fichier.](#)

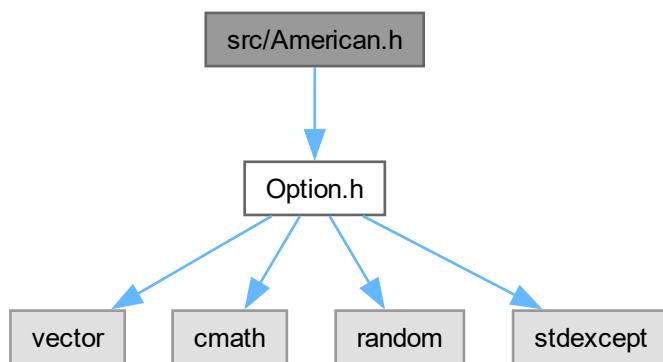
```
00001 #ifndef AGGREGATOR_H
00002 #define AGGREGATOR_H
00003
00004 #include <cmath>
00005 #include <stdexcept>
00006
00007 namespace crr {
00008
00012     class Aggregator {
00013     public:
00021         virtual double operator() (double agg, double price, double step) const = 0;
00022     };
00023
00027     class Arithmetic : public Aggregator {
00028     public:
00029         double operator() (double agg, double price, double step) const override {
00030             return (agg * step + price) / (step + 1);
00031         }
00032     };
00033
00037     class Geometric : public Aggregator {
00038     public:
00039         double operator() (double agg, double price, double step) const override {
00040             return std::pow(std::pow(agg, step) * price, 1.0 / (step + 1));
00041         }
00042     };
00043
00047     class LookMax : public Aggregator {
00048     public:
00049         double operator() (double agg, double price, double step) const override {
00050             return std::max<double>(agg, price);
00051         }
00052     };
00053
00057     class LookMin : public Aggregator {
00058     public:
00059         double operator() (double agg, double price, double step) const override {
```

```
00060         return std::min<double>(agg, price);
00061     }
00062 };
00063
00064 } // namespace crr
00065
00066 #endif // AGGREGATOR_H
```

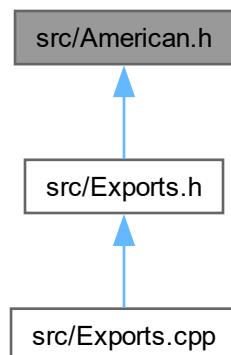
7.3 Référence du fichier src/American.h

#include "Option.h"

Graphe des dépendances par inclusion de American.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class `crr::American< TPayoff >`

Option américaine CRR avec couverture via décomposition de Doob.

Espaces de nommage

— namespace `crr`

7.4 American.h

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef AMERICAN_H
00002 #define AMERICAN_H
00003
00004 #include "Option.h"
00005
00006 namespace crr {
00007
00012     template<typename TPayoff>
00013     class American : public Option {
00014     private:
00015         TPayoff payoff_;
00016         std::vector<std::vector<double>> stockTree_;
00017         void buildStockTree();
00018
00019     public:
00020         American(double S0, double R, double sigma, double T, int N, const TPayoff& payoff);
00021         std::vector<std::vector<double>> treePrice() const override;
00022         HedgingStrategy hedgingStrategy() const override;
00023
00028         double priceRR() const;
00029
00034         double deltaRR() const;
00035     };
00036
00037     template<typename TPayoff>
00038     American<TPayoff>::American(double S0, double R, double sigma,
00039         double T, int N, const TPayoff& payoff)
00040         : Option(S0, R, sigma, T, N), payoff_(payoff)
00041     {
00042         buildStockTree();
00043     }
00044
00045     template<typename TPayoff>
00046     void American<TPayoff>::buildStockTree() {
00047         stockTree_.assign(N_ + 1, {});
00048         for (int n = 0; n <= N_; ++n) {
00049             stockTree_[n].resize(n + 1);
00050             for (int i = 0; i <= n; ++i) {
00051                 stockTree_[n][i] = S0_ * std::pow(1 + u_, i) * std::pow(1 + d_, n - i);
00052             }
00053         }
00054     }
00055
00056     template<typename TPayoff>
00057     std::vector<std::vector<double>> American<TPayoff>::treePrice() const {
00058         std::vector<std::vector<double>> V(N_ + 1);
00059         V[N_].resize(N_ + 1);
00060         for (int i = 0; i <= N_; ++i)
00061             V[N_][i] = payoff_(stockTree_[N_][i]);
00062
00063         for (int n = N_ - 1; n >= 0; --n) {
00064             V[n].resize(n + 1);
00065             for (int i = 0; i <= n; ++i) {
00066                 double cont = discount_ * 0.5 * (V[n + 1][i + 1] + V[n + 1][i]);
00067                 double exer = payoff_(stockTree_[n][i]);
00068                 V[n][i] = std::max<double>(exer, cont);
00069             }
00070         }
00071     }
00072 }
```

```

00073
00074     template<typename TPayoff>
00075     Option::HedgingStrategy American<TPayoff>::hedgingStrategy() const {
00076         // 0) Construction de l'arbre non recombinant
00077         std::vector<std::vector<double>> stockTreeNR;
00078         int N = N_;
00079         stockTreeNR.assign(N + 1, {});
00080         stockTreeNR[0] = { S0_ };
00081         for (int n = 1; n <= N; ++n) {
00082             int sz = 1 << n;
00083             stockTreeNR[n].resize(sz);
00084             for (int j = 0; j < sz; ++j) {
00085                 bool up = (j & 1);
00086                 double prev = stockTreeNR[n - 1][j >> 1];
00087                 stockTreeNR[n][j] = prev * (1 + (up ? u_ : d_));
00088             }
00089         }
00090
00091         // 1) Calcul de VNR sur arbre non recombinant
00092         std::vector<std::vector<double>> VNR(N + 1);
00093         VNR[N].resize(1 << N);
00094         for (int j = 0; j < (1 << N); ++j)
00095             VNR[N][j] = payoff_(stockTreeNR[N][j]);
00096
00097         for (int n = N - 1; n >= 0; --n) {
00098             int sz = 1 << n;
00099             VNR[n].resize(sz);
00100            for (int j = 0; j < sz; ++j) {
00101                double cont = discount_ * 0.5 * (VNR[n + 1][2 * j] + VNR[n + 1][2 * j + 1]);
00102                VNR[n][j] = std::max<double>(payoff_(stockTreeNR[n][j]), cont);
00103            }
00104        }
00105
00106         // 2) Calcul des incréments de compensateur
00107         std::vector<std::vector<double>> incr(N);
00108         for (int n = 0; n < N; ++n) {
00109             int sz = 1 << n;
00110             incr[n].resize(sz);
00111             for (int j = 0; j < sz; ++j) {
00112                 double cont = discount_ * 0.5 * (VNR[n + 1][2 * j] + VNR[n + 1][2 * j + 1]);
00113                 incr[n][j] = VNR[n][j] - cont;
00114             }
00115         }
00116
00117         // 3) Propagation forward pour A et calcul de M
00118         std::vector<std::vector<double>> A(N + 1), M(N + 1);
00119         A[0].resize(1);
00120         M[0].resize(1);
00121         A[0][0] = 0;
00122         M[0][0] = VNR[0][0];
00123         for (int n = 1; n <= N; ++n) {
00124             int sz = 1 << n;
00125             A[n].resize(sz);
00126             M[n].resize(sz);
00127             for (int j = 0; j < sz; ++j) {
00128                 double prevA = A[n - 1][j >> 1]; // j/2 pour la Fn-1 mesurabilité de An
00129                 double inc = incr[n - 1][j >> 1];
00130                 A[n][j] = (prevA + inc) / discount_; // faut confronter avec Vn, valeurs de An-1 e
00131                 incr dans n-1
00132                 M[n][j] = VNR[n][j] + A[n][j];
00133             }
00134         }
00135
00136         // 4) Couverture sur la martingale M
00137         HedgingStrategy H;
00138         H.delta.resize(N);
00139         H.bond.resize(N);
00140         for (int n = 0; n < N; ++n) {
00141             int sz = 1 << n;
00142             H.delta[n].resize(sz);
00143             H.bond[n].resize(sz);
00144             for (int j = 0; j < sz; ++j) {
00145                 double Mu = M[n + 1][2 * j + 1];
00146                 double Md = M[n + 1][2 * j];
00147                 double Su = stockTreeNR[n + 1][2 * j + 1];
00148                 double Sd = stockTreeNR[n + 1][2 * j];
00149                 double dlt = (Mu - Md) / (Su - Sd);
00150                 H.delta[n][j] = dlt;
00151                 H.bond[n][j] = M[n][j] - dlt * stockTreeNR[n][j];
00152             }
00153         }
00154         return H;
00155     }
00156
00157     template<typename TPayoff>
00158     double American<TPayoff>::priceRR() const {
00159         std::vector<int> Ns = { 100, 200, 400 };

```

```

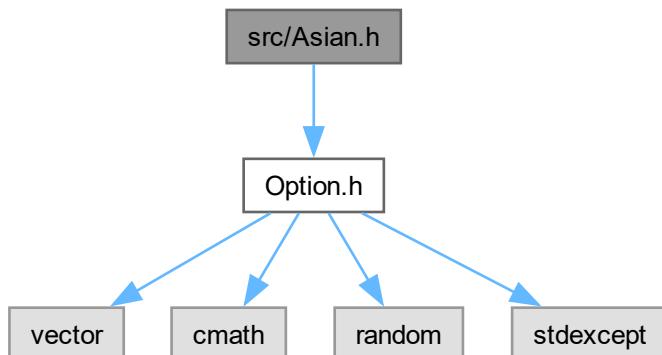
00159     int n = Ns.size(), m = 2;
00160     std::vector<std::vector<double>> A(n, std::vector<double>(m + 1));
00161     for (int i = 0; i < n; ++i) {
00162         American<TPayoff> opt(S0_, R_, sigma_, T_, Ns[i], payoff_);
00163         A[i][0] = opt.price();
00164     }
00165
00166     for (int k = 1; k <= m; ++k) {
00167         for (int i = 0; i + k < n; ++i) {
00168             double ratio = Ns[i + k] / Ns[i];
00169             A[i][k] = A[i + 1][k - 1] + (A[i + 1][k - 1] - A[i][k - 1]) / (ratio - 1.0);
00170         }
00171     }
00172     return A[0][m];
00173 }
00174
00175 template<typename TPayoff>
00176 double American<TPayoff>::deltaRR() const {
00177     double eps = 1e-4 * S0_;
00178     American<TPayoff> up(S0_ + eps, R_, sigma_, T_, N_, payoff_);
00179     American<TPayoff> dn(S0_ - eps, R_, sigma_, T_, N_, payoff_);
00180     double priceUp = up.priceRR();
00181     double priceDn = dn.priceRR();
00182     return (priceUp - priceDn) / (2 * eps);
00183 }
00184
00185 } // namespace crr
00186
00187 #endif // AMERICAN_H
00188

```

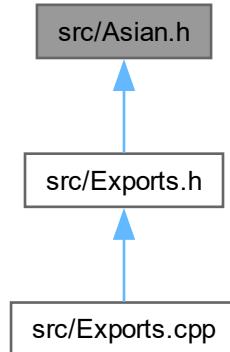
7.5 Référence du fichier src/Asian.h

#include "Option.h"

Graphe des dépendances par inclusion de Asian.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

- class `crr::Asian< TPayoff, TAggregator >`

Options path-dépendante CRR.

Espaces de nommage

- namespace `crr`

7.6 Asian.h

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef ASIAN_H
00002 #define ASIAN_H
00003
00004 #include "Option.h"
00005
00006 namespace crr {
00007
00013     template<typename TPayoff, typename TAggregator>
00014     class Asian : public Option {
00015     private:
00016         TPayoff payoff_;
00017         TAggregator aggregator_;
00018         std::vector<std::vector<double>> stockTreeNR_;
00019         void buildStockTreeNR();
00020
00021     public:
00022         Asian(double S0, double R, double sigma, double T, int N, const TPayoff& payoff, const
00023               TAggregator& aggregator);
00023             std::vector<std::vector<double>> treePrice() const override;
00024             HedgingStrategy hedgingStrategy() const override;
00025
00030             std::vector<double> terminalValues() const;
00031
00036             double priceMC() const;
00037
  
```

```

00042     double deltaMC() const;
00043 };
00044
00045 template<typename TPayoff, typename TAggregator>
00046 Asian<TPayoff, TAggregator>::Asian(double S0, double R, double sigma, double T,
00047     int N, const TPayoff& payoff, const TAggregator& aggregator)
00048 : Option(S0, R, sigma, T, N), payoff_(payoff), aggregator_(aggregator)
00049 {
00050     buildStockTreeNR();
00051 }
00052
00053 template<typename TPayoff, typename TAggregator>
00054 void Asian<TPayoff, TAggregator>::buildStockTreeNR() {
00055     stockTreeNR_.assign(N_ + 1, {});
00056     stockTreeNR_[0] = { S0_ };
00057     for (int n = 1; n <= N_; ++n) {
00058         int sz = 1 << n; // 2^n
00059         stockTreeNR_[n].resize(sz);
00060         for (int j = 0; j < sz; ++j) {
00061             bool up = (j & 1); // up = 1 si j impair, 0 sinon
00062             double prev = stockTreeNR_[n - 1][j >> 1]; // j/2
00063             stockTreeNR_[n][j] = prev * (1 + (up ? u_ : d_));
00064         }
00065     }
00066 }
00067
00068 template<typename TPayoff, typename TAggregator>
00069 std::vector<double> Asian<TPayoff, TAggregator>::terminalValues() const {
00070     int leafSz = 1 << N_;
00071     std::vector<double> vals(leafSz);
00072     for (int j = 0; j < leafSz; ++j) {
00073         double agg = S0_;
00074         for (int n = 1; n <= N_; ++n) {
00075             int idx = j >> (N_ - n);
00076             agg = aggregator_(agg, stockTreeNR_[n][idx], n);
00077         }
00078         vals[j] = payoff_(agg);
00079     }
00080     return vals;
00081 }
00082
00083 template<typename TPayoff, typename TAggregator>
00084 std::vector<std::vector<double>> Asian<TPayoff, TAggregator>::treePrice() const {
00085     auto terminal = terminalValues();
00086     std::vector<std::vector<double>> V(N_ + 1);
00087     int leafSz = 1 << N_;
00088     V[N_].resize(leafSz);
00089     for (int j = 0; j < leafSz; ++j)
00090         V[N_][j] = terminal[j];
00091
00092     for (int n = N_ - 1; n >= 0; --n) {
00093         int sz = 1 << n;
00094         V[n].resize(sz);
00095         for (int j = 0; j < sz; ++j)
00096             V[n][j] = discount_ * 0.5 * (V[n + 1][2 * j + 1] + V[n + 1][2 * j]);
00097     }
00098     return V;
00099 }
00100
00101 template<typename TPayoff, typename TAggregator>
00102 Option::HedgingStrategy Asian<TPayoff, TAggregator>::hedgingStrategy() const {
00103     auto V = treePrice();
00104     HedgingStrategy H;
00105     H.delta.resize(N_);
00106     H.bond.resize(N_);
00107     for (int n = 0; n < N_; ++n) {
00108         int sz = 1 << n;
00109         H.delta[n].resize(sz);
00110         H.bond[n].resize(sz);
00111         for (int j = 0; j < sz; ++j) {
00112             double Vu = V[n + 1][2 * j + 1];
00113             double Vd = V[n + 1][2 * j];
00114             double Su = stockTreeNR_[n + 1][2 * j + 1];
00115             double Sd = stockTreeNR_[n + 1][2 * j];
00116             double dlt = (Vu - Vd) / (Su - Sd);
00117             H.delta[n][j] = dlt;
00118             H.bond[n][j] = V[n][j] - dlt * stockTreeNR_[n][j];
00119         }
00120     }
00121     return H;
00122 }
00123
00124 template<typename TPayoff, typename TAggregator>
00125 double Asian<TPayoff, TAggregator>::priceMC() const {
00126     int steps = 100;
00127     int paths = 10000;
00128     std::mt19937_64 rng(42);

```

```

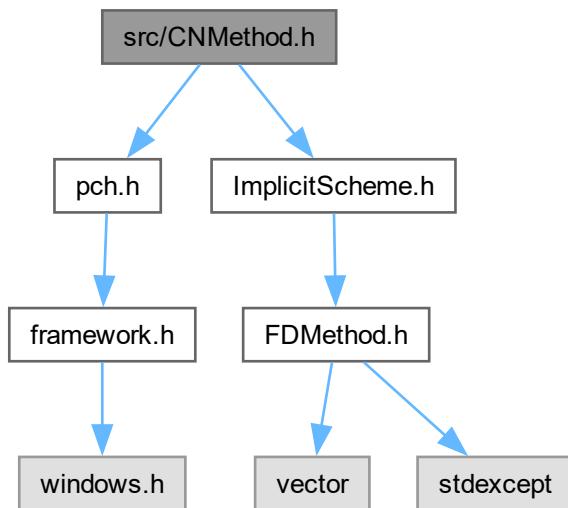
00129     std::normal_distribution<double> nd(0.0, 1.0);
00130     double dt = T_ / steps;
00131     double discount = std::exp(-R_ * T_);
00132     double sumPayoff = 0.0;
00133     for (int i = 0; i < paths; ++i) {
00134         double St = S0_;
00135         double agg = S0_;
00136         for (int j = 0; j < steps; ++j) {
00137             double Z = nd(rng);
00138             St *= std::exp((R_ - 0.5 * sigma_ * sigma_) * dt + sigma_ * std::sqrt(dt) * Z);
00139             agg = aggregator_(agg, St, j + 1);
00140         }
00141         sumPayoff += payoff_(agg);
00142     }
00143     return discount * sumPayoff / paths;
00144 }
00145
00146 template<typename TPayoff, typename TAggregator>
00147 double Asian<TPayoff, TAggregator>::deltaMC() const {
00148     double eps = 1e-4 * S0_;
00149     Asian<TPayoff, TAggregator> up(S0_ + eps, R_, sigma_, T_, N_, payoff_, aggregator_);
00150     Asian<TPayoff, TAggregator> dn(S0_ - eps, R_, sigma_, T_, N_, payoff_, aggregator_);
00151     double priceUp = up.priceMC();
00152     double priceDn = dn.priceMC();
00153     return (priceUp - priceDn) / (2 * eps);
00154 }
00155
00156 } // namespace crr
00157
00158 #endif // ASIAN_H
00159
00160

```

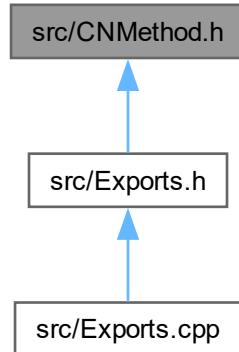
7.7 Référence du fichier src/CNMethod.h

```
#include "pch.h"
#include "ImplicitScheme.h"
```

Graphe des dépendances par inclusion de CNMethod.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

- class [pde::CNMethod< TPDE >](#)

Méthode de Crank–Nicolson pour le schéma implicite aux différences finies.

Espaces de nommage

- namespace [pde](#)

Macros

- [#define CNMETHOD_H](#)

7.7.1 Documentation des macros

7.7.1.1 CNMETHOD_H

```
#define CNMETHOD_H
```

7.8 CNMethod.h

[Aller à la documentation de ce fichier.](#)

```

00001 #include "pch.h"
00002
00003 #ifndef CNMETHOD_H
00004 #define CNMETHOD_H
00005
00006 #include "ImplicitScheme.h"
00007
00008 namespace pde {
00009
00014     template<typename TPDE>
00015     class CNMethod : public ImplicitScheme<TPDE> {
00016     public:
00017         CNMethod(const TPDE& pde, int imax, int jmax)
00018             : ImplicitScheme<TPDE>(pde, imax, jmax)
00019         {
00020         }
00021
00022         double A(int i, int j) const override {
00023             return 0.5 * this->dt_ * (this->b(i - 0.5, j) / 2.0 - this->a(i - 0.5, j) / this->ds_) /
00024             this->ds_;
00025
00026         double B(int i, int j) const override {
00027             return 1.0 + 0.5 * this->dt_ * (2.0 * this->a(i - 0.5, j) / (this->ds_ * this->ds_) -
00028             this->c(i - 0.5, j));
00029
00030         double C(int i, int j) const override {
00031             return -0.5 * this->dt_ * (this->b(i - 0.5, j) / 2.0 + this->a(i - 0.5, j) / this->ds_) /
00032             this->ds_;
00033
00034         double D(int i, int j) const override {
00035             return -this->dt_ * this->d(i - 0.5, j);
00036         }
00037
00038         double E(int i, int j) const override {
00039             return -A(i, j);
00040         }
00041
00042         double F(int i, int j) const override {
00043             return 2.0 - B(i, j);
00044         }
00045
00046         double G(int i, int j) const override {
00047             return -C(i, j);
00048         }
00049     };
00050
00051 } // namespace pde
00052
00053 #endif // CNMETHOD_H

```

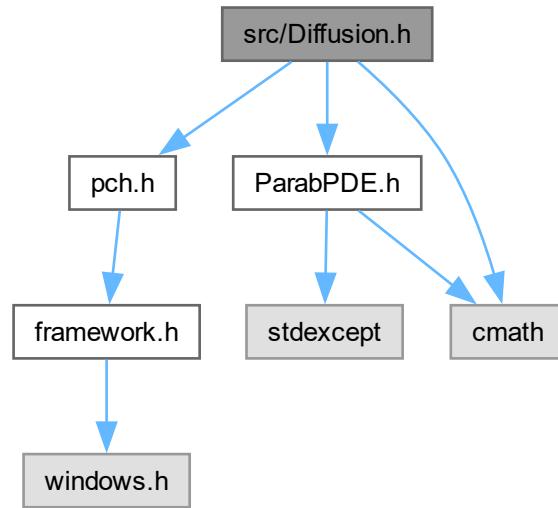
7.9 Référence du fichier src/Diffusion.h

```

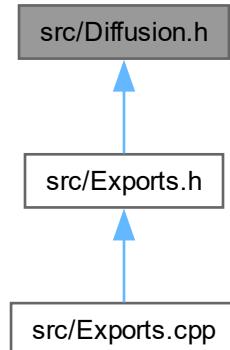
#include "pch.h"
#include "ParabPDE.h"
#include <cmath>

```

Graphe des dépendances par inclusion de Diffusion.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class [pde::Diffusion< TPayoff, TVol >](#)

EDP d'un modèle de diffusion.

Espaces de nommage

— namespace pde

7.10 Diffusion.h

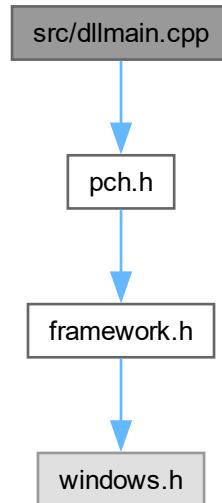
[Aller à la documentation de ce fichier.](#)

```
00001 #include "pch.h"
00002
00003 #ifndef DIFFUSION_H
00004 #define DIFFUSION_H
00005
00006 #include "ParabPDE.h"
00007 #include <cmath>
00008
00009 namespace pde {
00010
00016     template<typename TPayoff, typename TVol>
00017     class Diffusion : public ParabPDE {
00018     private:
00019         TPayoff payoff_;
00020         TVol vol_;
00021         double R_;
00022
00023     public:
00024         Diffusion(double T, double Smin, double Smax, double R, const TPayoff& payoff, const TVol&
00025           vol)
00026             : ParabPDE(T, Smin, Smax), R_(R), payoff_(payoff), vol_(vol)
00027         {
00028
00029             double a(double t, double S) const override {
00030                 return -0.5 * std::pow(vol_(t, S) * S, 2.0);
00031             }
00032
00033             double b(double t, double S) const override {
00034                 return -R_ * S;
00035             }
00036
00037             double c(double t, double S) const override {
00038                 return R_;
00039             }
00040
00041             double d(double t, double S) const override {
00042                 return 0;
00043             }
00044
00045             double Terminal(double S) const override {
00046                 return payoff_(S);
00047             }
00048
00049             double Lower(double t) const override {
00050                 return payoff_(Smin_) * std::exp(-R_ * (T_ - t));
00051             }
00052
00053             double Upper(double t) const override {
00054                 return payoff_(Smax_) * std::exp(-R_ * (T_ - t));
00055             }
00056     };
00057
00058 } // namespace pde
00059
00060 #endif // DIFFUSION_H
```

7.11 Référence du fichier src/dllmain.cpp

```
#include "pch.h"
```

Graphe des dépendances par inclusion de dllmain.cpp:



Fonctions

- BOOL APIENTRY [DlIMain](#) (HMODULE hModule, DWORD ul_reason_for_call, LPVOID lpReserved)

7.11.1 Documentation des fonctions

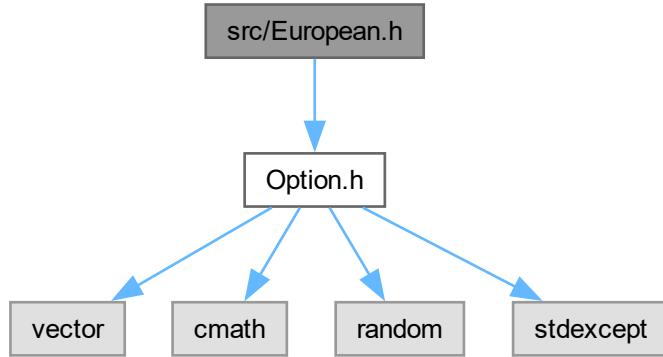
7.11.1.1 [DlIMain\(\)](#)

```
BOOL APIENTRY DlIMain (
    HMODULE hModule,
    DWORD ul_reason_for_call,
    LPVOID lpReserved)
```

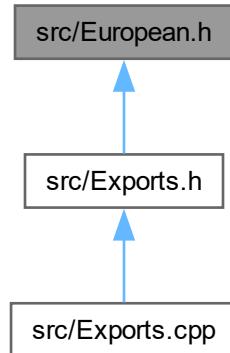
7.12 Référence du fichier src/European.h

```
#include "Option.h"
```

Graphe des dépendances par inclusion de European.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

- class [crr::European< TPayoff >](#)

Option européenne CRR.

Espaces de nommage

- namespace [crr](#)

7.13 European.h

[Aller à la documentation de ce fichier.](#)

```

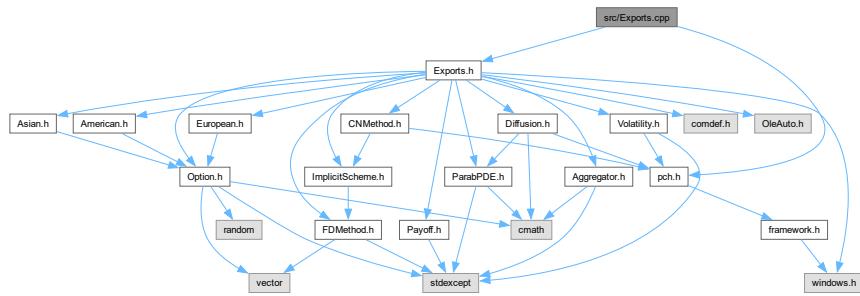
00001 #ifndef EUROPEAN_H
00002 #define EUROPEAN_H
00003
00004 #include "Option.h"
00005
00006 namespace crr {
00007
00012     template<typename TPayoff>
00013     class European : public Option {
00014     private:
00015         TPayoff payoff_;
00016         std::vector<std::vector<double>> stockTree_;
00017         void buildStockTree();
00018
00019     public:
00020         European(double S0, double R, double sigma, double T, int N, const TPayoff& payoff);
00021         std::vector<std::vector<double>> treePrice() const override;
00022         HedgingStrategy hedgingStrategy() const override;
00023     };
00024
00025     template<typename TPayoff>
00026     European<TPayoff>::European(double S0, double R, double sigma,
00027         double T, int N, const TPayoff& payoff)
00028         : Option(S0, R, sigma, T, N), payoff_(payoff)
00029     {
00030         buildStockTree();
00031     }
00032
00033     template<typename TPayoff>
00034     void European<TPayoff>::buildStockTree() {
00035         stockTree_.assign(N_ + 1, {});
00036         for (int n = 0; n <= N_; ++n) {
00037             stockTree_[n].resize(n + 1);
00038             for (int i = 0; i <= n; ++i) {
00039                 stockTree_[n][i] = S0_ * std::pow(1 + u_, i) * std::pow(1 + d_, n - i);
00040             }
00041         }
00042     }
00043
00044     template<typename TPayoff>
00045     std::vector<std::vector<double>> European<TPayoff>::treePrice() const {
00046         std::vector<std::vector<double>> V(N_ + 1);
00047         V[N_].resize(N_ + 1);
00048         // Valeurs terminales
00049         for (int i = 0; i <= N_; ++i)
00050             V[N_][i] = payoff_(stockTree_[N_][i]);
00051
00052         // Rétropropagation
00053         for (int n = N_ - 1; n >= 0; --n) {
00054             V[n].resize(n + 1);
00055             for (int i = 0; i <= n; ++i) {
00056                 V[n][i] = discount_ * 0.5 * (V[n + 1][i + 1] + V[n + 1][i]);
00057             }
00058         }
00059         return V;
00060     }
00061
00062     template<typename TPayoff>
00063     Option::HedgingStrategy European<TPayoff>::hedgingStrategy() const {
00064         auto V = treePrice();
00065         HedgingStrategy H;
00066         H.delta.resize(N_);
00067         H.bond.resize(N_);
00068         for (int n = 0; n < N_; ++n) {
00069             H.delta[n].resize(n + 1);
00070             H.bond[n].resize(n + 1);
00071             for (int i = 0; i <= n; ++i) {
00072                 double Vu = V[n + 1][i + 1];
00073                 double Vd = V[n + 1][i];
00074                 double Su = stockTree_[n + 1][i + 1];
00075                 double Sd = stockTree_[n + 1][i];
00076                 double dlt = (Vu - Vd) / (Su - Sd);
00077                 H.delta[n][i] = dlt;
00078                 H.bond[n][i] = V[n][i] - dlt * stockTree_[n][i];
00079             }
00080         }
00081         return H;
00082     }
00083
00084 } // namespace crr
00085
00086 #endif // EUROPEAN_H

```

00087

7.14 Référence du fichier src/Exports.cpp

```
#include "pch.h"
#include "Exports.h"
Graphe des dépendances par inclusion de Exports.cpp:
```



Macros

- #define **SAFE_DOUBLE**(name, args, body)

Déclare une fonction extern "C" __stdcall retournant un double protégée par try/catch.

- #define **SAFE_VARIANT**(name, args, body)

Déclare une fonction extern "C" __stdcall retournant un VARIANT SAFEARRAY protégée par try/catch.

Définitions de type

- using **DiffusionPutBS** = pde::Diffusion<opt::PayoffPut, pde::BSVol>
- using **DiffusionDigitCallBS** = pde::Diffusion<opt::PayoffDigitCall, pde::BSVol>
- using **DiffusionDigitPutBS** = pde::Diffusion<opt::PayoffDigitPut, pde::BSVol>
- using **DiffusionDoubleDigitBS** = pde::Diffusion<opt::PayoffDoubleDigit, pde::BSVol>
- using **DiffusionBullBS** = pde::Diffusion<opt::PayoffBull, pde::BSVol>
- using **DiffusionBearBS** = pde::Diffusion<opt::PayoffBear, pde::BSVol>
- using **DiffusionStrangleBS** = pde::Diffusion<opt::PayoffStrangle, pde::BSVol>
- using **DiffusionButterflyBS** = pde::Diffusion<opt::PayoffButterfly, pde::BSVol>
- using **DiffusionPutVL** = pde::Diffusion<opt::PayoffPut, pde::LocalVol>
- using **DiffusionDigitCallVL** = pde::Diffusion<opt::PayoffDigitCall, pde::LocalVol>
- using **DiffusionDigitPutVL** = pde::Diffusion<opt::PayoffDigitPut, pde::LocalVol>

- using `DiffusionDoubleDigitVL = pde::Diffusion<opt::PayoffDoubleDigit, pde::LocalVol>`
- using `DiffusionBullVL = pde::Diffusion<opt::PayoffBull, pde::LocalVol>`
- using `DiffusionBearVL = pde::Diffusion<opt::PayoffBear, pde::LocalVol>`
- using `DiffusionStrangleVL = pde::Diffusion<opt::PayoffStrangle, pde::LocalVol>`
- using `DiffusionButterflyVL = pde::Diffusion<opt::PayoffButterfly, pde::LocalVol>`

Fonctions

- `__declspec (dllexport) void __stdcall ResetErrorFlag()`

Calcule le delta d'un call européen.
- `double reportError (const char *msg, const char *title)`

Affiche une boîte d'erreur une seule fois.
- `VARIANT makeVariantFromArray (SAFEARRAY *psa)`

Emballer un SAFEARRAY de doubles dans un VARIANT.*
- template<typename Mtx>
`VARIANT toVariant (const Mtx &matrix)`

Convertit un conteneur matriciel en VARIANT COM pour l'interopérabilité.
- `SAFE_DOUBLE (PriceEuCall,(double S0, double R, double sigma, double T, int N, double K), return crr::European< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).price();) SAFE_DOUBLE(Delta← EuCall`
- `double double double int double return crr::European< opt::PayoffCall > (S0, R, sigma, T, N, opt::PayoffCall(K)).deltaZero()`
- `SAFE_VARIANT (TreeEuCall,(double S0, double R, double sigma, double T, int N, double K), { auto V=crr::European< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).treePrice();return toVariant(V);}) SAFE_VARIANT(TreeDeltaEuCall`
- `double double double int double return toVariant (H.delta)`
- `SAFE_VARIANT (TreeBondEuCall,(double S0, double R, double sigma, double T, int N, double K), { auto H=crr::European< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).hedgingStrategy();return toVariant(H.bond);}) SAFE_DOUBLE(PriceEuPut`
- `double double double int double return crr::European< opt::PayoffPut > (S0, R, sigma, T, N, opt::PayoffPut(K)).price()`
- `SAFE_DOUBLE (DeltaEuPut,(double S0, double R, double sigma, double T, int N, double K), return crr::European< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).deltaZero();) SAFE_VARIANT(Tree← EuPut`
- `double double double int double return toVariant (V)`
- `SAFE_VARIANT (TreeDeltaEuPut,(double S0, double R, double sigma, double T, int N, double K), { auto H=crr::European< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).hedgingStrategy();return toVariant(H.delta);}) SAFE_VARIANT(TreeBondEuPut`

- `SAFE_DOUBLE` (`PriceDigitCall(double S0, double R, double sigma, double T, int N, double K)`, return `crr::European< opt::PayoffDigitCall >(S0, R, sigma, T, N, opt::PayoffDigitCall(K)).price();`) `SAFE_DOUBLE(DeltaDigitCall`
- `double double double double int double return crr::European< opt::PayoffDigitCall > (S0, R, sigma, T, N, opt::PayoffDigitCall(K)).deltaZero()`
- `SAFE_VARIANT` (`TreeDigitCall(double S0, double R, double sigma, double T, int N, double K)`, { auto `V=crr::European< opt::PayoffDigitCall >(S0, R, sigma, T, N, opt::PayoffDigitCall(K)).treePrice();`return `toVariant(V);`}) `SAFE_VARIANT(TreeDeltaDigitCall`
- `SAFE_VARIANT` (`TreeBondDigitCall(double S0, double R, double sigma, double T, int N, double K)`, { auto `H=crr::European< opt::PayoffDigitCall >(S0, R, sigma, T, N, opt::PayoffDigitCall(K)).hedgingStrategy();`return `toVariant(H.bond);`}) `SAFE_DOUBLE(PriceDigitPut`
- `double double double double int double return crr::European< opt::PayoffDigitPut > (S0, R, sigma, T, N, opt::PayoffDigitPut(K)).price()`
- `SAFE_DOUBLE` (`DeltaDigitPut(double S0, double R, double sigma, double T, int N, double K)`, return `crr::European< opt::PayoffDigitPut >(S0, R, sigma, T, N, opt::PayoffDigitPut(K)).deltaZero();`) `SAFE_VARIANT(TreeDigitPut`
- `SAFE_VARIANT` (`TreeDeltaDigitPut(double S0, double R, double sigma, double T, int N, double K)`, { auto `H=crr::European< opt::PayoffDigitPut >(S0, R, sigma, T, N, opt::PayoffDigitPut(K)).hedgingStrategy();`return `toVariant(H.delta);`}) `SAFE_VARIANT(TreeBondDigitPut`
- `SAFE_DOUBLE` (`PriceDD(double S0, double R, double sigma, double T, int N, double K1, double K2)`, return `crr::European< opt::PayoffDoubleDigit >(S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1, K2)).price();`) `SAFE_DOUBLE(DeltaDD`
- `double double double double int double double return crr::European< opt::PayoffDoubleDigit > (S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1, K2)).deltaZero()`
- `SAFE_VARIANT` (`TreeDD(double S0, double R, double sigma, double T, int N, double K1, double K2)`, { auto `V=crr::European< opt::PayoffDoubleDigit >(S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1, K2)).treePrice();`return `toVariant(V);`}) `SAFE_VARIANT(TreeDeltaDD`
- `SAFE_VARIANT` (`TreeBondDD(double S0, double R, double sigma, double T, int N, double K1, double K2)`, { auto `H=crr::European< opt::PayoffDoubleDigit >(S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1, K2)).hedgingStrategy();`return `toVariant(H.bond);`}) `SAFE_DOUBLE(PriceBull`
- `double double double double int double double return crr::European< opt::PayoffBull > (S0, R, sigma, T, N, opt::PayoffBull(K1, K2)).price()`
- `SAFE_DOUBLE` (`DeltaBull(double S0, double R, double sigma, double T, int N, double K1, double K2)`, return `crr::European< opt::PayoffBull >(S0, R, sigma, T, N, opt::PayoffBull(K1, K2)).deltaZero();`) `SAFE_VARIANT(TreeBull`
- `SAFE_VARIANT` (`TreeDeltaBull(double S0, double R, double sigma, double T, int N, double K1, double K2)`, { auto `H=crr::European< opt::PayoffBull >(S0, R, sigma, T, N, opt::PayoffBull(K1, K2)).hedgingStrategy();`return `toVariant(H.delta);`}) `SAFE_VARIANT(TreeBondBull`
- `SAFE_DOUBLE` (`PriceBear(double S0, double R, double sigma, double T, int N, double K1, double K2)`, return `crr::European< opt::PayoffBear >(S0, R, sigma, T, N, opt::PayoffBear(K1, K2)).price();`) `SAFE_DOUBLE(DeltaBear`
- `double double double double int double double return crr::European< opt::PayoffBear > (S0, R, sigma, T, N, opt::PayoffBear(K1, K2)).deltaZero()`
- `SAFE_VARIANT` (`TreeBear(double S0, double R, double sigma, double T, int N, double K1, double K2)`, { auto `V=crr::European< opt::PayoffBear >(S0, R, sigma, T, N, opt::PayoffBear(K1, K2)).treePrice();`return

- `toVariant(V);}) SAFE_VARIANT(TreeDeltaBear`
- `SAFE_VARIANT (TreeBondBear,(double S0, double R, double sigma, double T, int N, double K1, double K2), { auto H=crr::European< opt::PayoffBear >(S0, R, sigma, T, N, opt::PayoffBear(K1, K2)).hedging←Strategy();return toVariant(H.bond);}) SAFE_DOUBLE(PriceStrangle`
- `double double double double int double double return crr::European< opt::PayoffStrangle > (S0, R, sigma, T, N, opt::PayoffStrangle(K1, K2)).price()`
- `SAFE_DOUBLE (DeltaStrangle,(double S0, double R, double sigma, double T, int N, double K1, double K2), return crr::European< opt::PayoffStrangle >(S0, R, sigma, T, N, opt::PayoffStrangle(K1, K2)).deltaZero();) SAFE_VARIANT(TreeStrangle`
- `SAFE_VARIANT (TreeDeltaStrangle,(double S0, double R, double sigma, double T, int N, double K1, double K2), { auto H=crr::European< opt::PayoffStrangle >(S0, R, sigma, T, N, opt::PayoffStrangle(K1, K2)).hedgingStrategy();return toVariant(H.delta);}) SAFE_VARIANT(TreeBondStrangle`
- `SAFE_DOUBLE (PriceButterfly,(double S0, double R, double sigma, double T, int N, double K1, double K2), return crr::European< opt::PayoffButterfly >(S0, R, sigma, T, N, opt::PayoffButterfly(K1, K2)).price();) SAFE_DOUBLE(DeltaButterfly`
- `double double double double int double double return crr::European< opt::PayoffButterfly > (S0, R, sigma, T, N, opt::PayoffButterfly(K1, K2)).deltaZero()`
- `SAFE_VARIANT (TreeButterfly,(double S0, double R, double sigma, double T, int N, double K1, double K2), { auto V=crr::European< opt::PayoffButterfly >(S0, R, sigma, T, N, opt::PayoffButterfly(K1, K2)).tree←Price();return toVariant(V);}) SAFE_VARIANT(TreeDeltaButterfly`
- `SAFE_VARIANT (TreeBondButterfly,(double S0, double R, double sigma, double T, int N, double K1, double K2), { auto H=crr::European< opt::PayoffButterfly >(S0, R, sigma, T, N, opt::PayoffButterfly(K1, K2)).hedgingStrategy();return toVariant(H.bond);}) using AsianCallArithmetic`
- `SAFE_DOUBLE (PriceAritCall,(double S0, double R, double sigma, double T, int N, double K), return AsianCallArithmetic(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).price();) SAFE_DOUBLE(Delta←AritCall`
- `double double double double int double return AsianCallArithmetic (S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).deltaZero()`
- `SAFE_VARIANT (TreeAritCall,(double S0, double R, double sigma, double T, int N, double K), { auto V=AsianCallArithmetic(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).treePrice();return toVariant(V);}) SAFE_VARIANT(TreeDeltaAritCall`
- `SAFE_VARIANT (TreeBondAritCall,(double S0, double R, double sigma, double T, int N, double K), { auto H=AsianCallArithmetic(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).hedgingStrategy();return toVariant(H.bond);}) SAFE_DOUBLE(PriceAritCallMC`
- `SAFE_DOUBLE (DeltaAritCallMC,(double S0, double R, double sigma, double T, int N, double K), return AsianCallArithmetic(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).deltaMC();) using AsianPutArithmetic`
- `SAFE_DOUBLE (PriceAritPut,(double S0, double R, double sigma, double T, int N, double K), return AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).price();) SAFE_DOUBLE(Delta←AritPut`
- `double double double double int double return AsianPutArithmetic (S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).deltaZero()`
- `SAFE_VARIANT (TreeAritPut,(double S0, double R, double sigma, double T, int N, double K), { auto V=AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).treePrice();return toVariant(V);}) SAFE_VARIANT(TreeDeltaAritPut`

- `SAFE_VARIANT` (`TreeBondAritPut(double S0, double R, double sigma, double T, int N, double K)`, {
auto `H=AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).hedgingStrategy();return toVariant(H.bond);}`) `SAFE_DOUBLE`(`PriceAritPutMC`
- `SAFE_DOUBLE` (`DeltaAritPutMC(double S0, double R, double sigma, double T, int N, double K)`,
return `AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic().deltaMC();)` using
`AsianCallGeometric`
- `SAFE_DOUBLE` (`PriceGeomCall(double S0, double R, double sigma, double T, int N, double K)`, re-
turn `AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric().price();)`) `SAFE_DOUBLE`(`DeltaGeomCall`
- double double double double int double return `AsianCallGeometric (S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric().deltaZero()`
- `SAFE_VARIANT` (`TreeGeomCall(double S0, double R, double sigma, double T, int N, double K)`, {
auto `V=AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric().treePrice();return toVariant(V);}`) `SAFE_VARIANT`(`TreeDeltaGeomCall`
- `SAFE_VARIANT` (`TreeBondGeomCall(double S0, double R, double sigma, double T, int N, double K)`, {
auto `H=AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric().hedgingStrategy();return toVariant(H.bond);}`) `SAFE_DOUBLE`(`PriceGeomCallMC`
- `SAFE_DOUBLE` (`DeltaGeomCallMC(double S0, double R, double sigma, double T, int N, double K)`,
return `AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric().deltaMC();)` using
`AsianPutGeometric`
- `SAFE_DOUBLE` (`PriceGeomPut(double S0, double R, double sigma, double T, int N, double K)`, return
`AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric().price();)`) `SAFE_DOUBLE`(`DeltaGeomPut`
- double double double double int double return `AsianPutGeometric (S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric().deltaZero()`
- `SAFE_VARIANT` (`TreeGeomPut(double S0, double R, double sigma, double T, int N, double K)`, {
auto `V=AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric().treePrice();return toVariant(V);}`) `SAFE_VARIANT`(`TreeDeltaGeomPut`
- `SAFE_VARIANT` (`TreeBondGeomPut(double S0, double R, double sigma, double T, int N, double K)`, {
auto `H=AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric().hedgingStrategy();return toVariant(H.bond);}`) `SAFE_DOUBLE`(`PriceGeomPutMC`
- `SAFE_DOUBLE` (`DeltaGeomPutMC(double S0, double R, double sigma, double T, int N, double K)`,
return `AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric().deltaMC();)` using
`AsianCallLookMax`
- `SAFE_DOUBLE` (`PriceMaxCall(double S0, double R, double sigma, double T, int N, double K)`, return
`AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax().price();)`) `SAFE_DOUBLE`(`DeltaMaxCall`
- double double double double int double return `AsianCallLookMax (S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax().deltaZero()`
- `SAFE_VARIANT` (`TreeMaxCall(double S0, double R, double sigma, double T, int N, double K)`, { auto
`V=AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax().treePrice();return toVariant(V);}`) `SAFE_VARIANT`(`TreeDeltaMaxCall`
- `SAFE_VARIANT` (`TreeBondMaxCall(double S0, double R, double sigma, double T, int N, double K)`, {
auto `H=AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax().hedgingStrategy();return toVariant(H.bond);}`) `SAFE_DOUBLE`(`PriceMaxCallMC`

- `SAFE_DOUBLE` (`DeltaMaxCallMC(double S0, double R, double sigma, double T, int N, double K)`,
`return AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax()).deltaMC();`) using
`AsianPutLookMin`
- `SAFE_DOUBLE` (`PriceMinPut(double S0, double R, double sigma, double T, int N, double K)`,
`return AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin().price();`) `SAFE_DOUBLE(DeltaMinPut)`
- `double double double double int double return AsianPutLookMin (S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin()).deltaZero()`
- `SAFE_VARIANT` (`TreeMinPut(double S0, double R, double sigma, double T, int N, double K)`, { auto
`V=AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin().treePrice();return toVariant(V);})`
`SAFE_VARIANT(TreeDeltaMinPut`
- `SAFE_VARIANT` (`TreeBondMinPut(double S0, double R, double sigma, double T, int N, double K)`, { auto
`H=AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin().hedgingStrategy();return toVariant(H.bond);})` `SAFE_DOUBLE(PriceMinPutMC`
- `SAFE_DOUBLE` (`DeltaMinPutMC(double S0, double R, double sigma, double T, int N, double K)`,
`return AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin().deltaMC();`) `SAFE_DOUBLE(PriceAmCall`
- `double double double double int double return crr::American< opt::PayoffCall > (S0, R, sigma, T, N, opt::PayoffCall(K)).price()`
- `SAFE_DOUBLE` (`DeltaAmCall(double S0, double R, double sigma, double T, int N, double K)`,
`return crr::American< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K).deltaZero();`) `SAFE_VARIANT(TreeAmCall`
- `SAFE_VARIANT` (`TreeDeltaAmCall(double S0, double R, double sigma, double T, int N, double K)`, { auto
`H=crr::American< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K).hedgingStrategy();return toVariant(H.delta);})` `SAFE_VARIANT(TreeBondAmCall`
- `SAFE_DOUBLE` (`PriceAmPut(double S0, double R, double sigma, double T, int N, double K)`,
`return crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K).price();`) `SAFE_DOUBLE(DeltaAmPut`
- `double double double double int double return crr::American< opt::PayoffPut > (S0, R, sigma, T, N, opt::PayoffPut(K)).deltaZero()`
- `SAFE_VARIANT` (`TreeAmPut(double S0, double R, double sigma, double T, int N, double K)`, { auto
`V=crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K).treePrice();return toVariant(V);})`
`SAFE_VARIANT(TreeDeltaAmPut`
- `SAFE_VARIANT` (`TreeBondAmPut(double S0, double R, double sigma, double T, int N, double K)`, { auto
`H=crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K).hedgingStrategy();return toVariant(H.bond);})` `SAFE_DOUBLE(PriceAmPutRR`
- `SAFE_DOUBLE` (`DeltaAmPutRR(double S0, double R, double sigma, double T, int N, double K)`,
`return crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K).deltaRR();`) using `DiffusionCallIBS`
- `SAFE_DOUBLE` (`PriceEuCallIBS(double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax)`, { `DiffusionCallIBS eq(T, Smin, Smax, R, opt::PayoffCall(K, pde::BSVol(sigma));pde::CNMethod< DiffusionCallIBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);`}) `SAFE_DOUBLE(DeltaEuCallIBS`
- `double double double double double double int int pde::CNMethod< DiffusionCallIBS > solver (eq, imax, jmax)`
- `solver SolvePDE ()`

- return solver.delta(t, S)
- SAFE_VARIANT (GridPriceEuCallBS,(double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionCallBS eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::BSVol(sigma));pde::CNMethod< DiffusionCallBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);})
SAFE_VARIANT(GridDeltaEuCallBS)
- return toVariant(G.del)
- SAFE_DOUBLE (PriceEuPutBS,(double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionPutBS eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::BSVol(sigma));pde::CNMethod< DiffusionPutBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaEuPutBS)
- SAFE_VARIANT (GridPriceEuPutBS,(double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionPutBS eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::BSVol(sigma));pde::CNMethod< DiffusionPutBS > solver(eq, imax, jmax);solver.SolvePDE();return toVariant(G.val);})
SAFE_VARIANT(GridDeltaEuPutBS)
- SAFE_DOUBLE (PriceDigitCallBS,(double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitCallBS eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::BSVol(sigma));pde::CNMethod< DiffusionDigitCallBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaDigitCallBS)
- SAFE_VARIANT (GridPriceDigitCallBS,(double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitCallBS eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::BSVol(sigma));pde::CNMethod< DiffusionDigitCallBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaDigitCallBS)
- SAFE_DOUBLE (PriceDigitPutBS,(double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitPutBS eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::BSVol(sigma));pde::CNMethod< DiffusionDigitPutBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaDigitPutBS)
- SAFE_VARIANT (GridPriceDigitPutBS,(double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitPutBS eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::BSVol(sigma));pde::CNMethod< DiffusionDigitPutBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaDigitPutBS)
- SAFE_DOUBLE (PriceDDBS,(double t, double S, double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionDoubleDigitBS eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionDoubleDigitBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaDDBS)
- SAFE_VARIANT (GridPriceDDBS,(double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionDoubleDigitBS eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionDoubleDigitBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaDDBS)
- SAFE_DOUBLE (PriceBullBS,(double t, double S, double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBullBS eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionBullBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaBullBS)
- SAFE_VARIANT (GridPriceBullBS,(double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBullBS eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionBullBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaBullBS)
- SAFE_DOUBLE (PriceBearBS,(double t, double S, double sigma, double T, double R, double K1,

- ```

double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBearBS eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionBearBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaBearBS

— SAFE_VARIANT (GridPriceBearBS,double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBearBS eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionBearBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaBearBS

— SAFE_DOUBLE (PriceStrangleBS,double t, double S, double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionStrangleBS eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionStrangleBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaStrangleBS

— SAFE_VARIANT (GridPriceStrangleBS,double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionStrangleBS eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionStrangleBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaStrangleBS

— SAFE_DOUBLE (PriceButterflyBS,double t, double S, double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionButterflyBS eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionButterflyBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaButterflyBS

— SAFE_VARIANT (GridPriceButterflyBS,double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionButterflyBS eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::BSVol(sigma));pde::CNMethod< DiffusionButterflyBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaButterflyBS

— SAFE_DOUBLE (Vol,(double t, double S, double alfa, double beta), return pde::LocalVol(alfa, beta)(t, S);) using DiffusionCallVL

— SAFE_DOUBLE (PriceEuCallVL,(double t, double S, double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionCallVL eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionCallVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaEuCallVL

— SAFE_VARIANT (GridPriceEuCallVL,(double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionCallVL eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionCallVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaEuCallVL

— SAFE_DOUBLE (PriceEuPutVL,(double t, double S, double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionPutVL eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionPutVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaEuPutVL

— SAFE_VARIANT (GridPriceEuPutVL,(double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionPutVL eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionPutVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaEuPutVL

— SAFE_DOUBLE (PriceDigitCallVL,(double t, double S, double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitCallVL eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDigitCallVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaDigitCallVL

— SAFE_VARIANT (GridPriceDigitCallVL,(double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitCallVL eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDigitCallVL > solver(eq, imax, jmax);solver.SolvePDE();auto

```

- `G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaDigitCallVL`
- `SAFE_DOUBLE (PriceDigitPutVL,(double t, double S, double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitPutVL eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDigitPutVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaDigitPutVL`
- `SAFE_VARIANT (GridPriceDigitPutVL,(double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax, int jmax), { DiffusionDigitPutVL eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDigitPutVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaDigitPutVL`
- `SAFE_DOUBLE (PriceDDVL,(double t, double S, double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionDoubleDigitVL eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDoubleDigitVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaDDVL`
- `SAFE_VARIANT (GridPriceDDVL,(double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionDoubleDigitVL eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDoubleDigitVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaDDVL`
- `SAFE_DOUBLE (PriceBullVL,(double t, double S, double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBullVL eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionBullVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaBullVL`
- `SAFE_VARIANT (GridPriceBullVL,(double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBullVL eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionBullVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaBullVL`
- `SAFE_DOUBLE (PriceBearVL,(double t, double S, double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBearVL eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionBearVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaBearVL`
- `SAFE_VARIANT (GridPriceBearVL,(double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionBearVL eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionBearVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaBearVL`
- `SAFE_DOUBLE (PriceStrangleVL,(double t, double S, double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionStrangleVL eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionStrangleVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaStrangleVL`
- `SAFE_VARIANT (GridPriceStrangleVL,(double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionStrangleVL eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionStrangleVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDeltaStrangleVL`
- `SAFE_DOUBLE (PriceButterflyVL,(double t, double S, double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax, int imax, int jmax), { DiffusionButterflyVL eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionButterflyVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);}) SAFE_DOUBLE(DeltaButterflyVL`
- `SAFE_VARIANT (GridPriceButterflyVL,(double alfa, double beta, double T, double R, double K1, double`

```
K2, double Smin, double Smax, int imax, int jmax), { DiffusionButterflyVL eq(T, Smin, Smax, R,
opt::PayoffButterfly(K1, K2), pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionButterflyVL > solver(eq,
imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.val);}) SAFE_VARIANT(GridDelta<-
ButterflyVL
```

## Variables

- double **S0**
- double double **R**
- double double double **sigma**
- double double double double **T**
- double double double double int **N**
- double double double double int double **K**
- double double double double int double **K1**
- double double double double int double double **K2**
- double **t**
- double double **S**
- double double double double double double double **Smin**
- double double double double double double double double **Smax**
- double double double double double double double int **imax**
- double double double double double double double int int **jmax**
- auto **G** = solver.grid()
- double double double **alfa**
- double double double double **beta**

## 7.14.1 Documentation des macros

### 7.14.1.1 SAFE\_DOUBLE

```
#define SAFE_DOUBLE(
 name,
 args,
 body)
```

#### Valeur :

```
extern "C" double __stdcall name args {
 try { body; }
 catch(const std::exception& e) {
 return reportError(e.what(), #name);
 }
 catch(...) {
 return reportError("Erreur inconnue dans " #name, #name); \
 }
}
```

Déclare une fonction extern "C" \_\_stdcall retournant un double protégée par try/catch.

### Paramètres

|             |                                               |
|-------------|-----------------------------------------------|
| <i>name</i> | Identifiant de la fonction exportée.          |
| <i>args</i> | Signature (entre parenthèses) de la fonction. |
| <i>body</i> | Code à exécuter (doit inclure un return).     |

### 7.14.1.2 SAFE\_VARIANT

```
#define SAFE_VARIANT(
 name,
 args,
 body)
```

#### Valeur :

```
extern "C" VARIANT __stdcall name args {
 try { body; }
 catch(const std::exception& e) {
 reportError(e.what(), #name);
 return makeVariantFromArray(nullptr);
 }
 catch(...) {
 reportError("Erreur inconnue dans " #name, #name);
 return makeVariantFromArray(nullptr);
 }
}
```

Déclare une fonction extern "C" \_\_stdcall retournant un VARIANT SAFEARRAY protégée par try/catch.

### Paramètres

|             |                                                    |
|-------------|----------------------------------------------------|
| <i>name</i> | Identifiant de la fonction exportée.               |
| <i>args</i> | Signature (entre parenthèses) de la fonction.      |
| <i>body</i> | Bloc { ... } remplissant et retournant un VARIANT. |

## 7.14.2 Documentation des définitions de type

### 7.14.2.1 DiffusionBearBS

```
using DiffusionBearBS = pde::Diffusion<opt::PayoffBear, pde::BSVol>
```

### 7.14.2.2 DiffusionBearVL

```
using DiffusionBearVL = pde::Diffusion<opt::PayoffBear, pde::LocalVol>
```

### 7.14.2.3 DiffusionBullBS

```
using DiffusionBullBS = pde::Diffusion<opt::PayoffBull, pde::BSVol>
```

#### 7.14.2.4 DiffusionBullVL

```
using DiffusionBullVL = pde::Diffusion<opt::PayoffBull, pde::LocalVol>
```

#### 7.14.2.5 DiffusionButterflyBS

```
using DiffusionButterflyBS = pde::Diffusion<opt::PayoffButterfly, pde::BSVol>
```

#### 7.14.2.6 DiffusionButterflyVL

```
using DiffusionButterflyVL = pde::Diffusion<opt::PayoffButterfly, pde::LocalVol>
```

#### 7.14.2.7 DiffusionDigitCallBS

```
using DiffusionDigitCallBS = pde::Diffusion<opt::PayoffDigitCall, pde::BSVol>
```

#### 7.14.2.8 DiffusionDigitCallVL

```
using DiffusionDigitCallVL = pde::Diffusion<opt::PayoffDigitCall, pde::LocalVol>
```

#### 7.14.2.9 DiffusionDigitPutBS

```
using DiffusionDigitPutBS = pde::Diffusion<opt::PayoffDigitPut, pde::BSVol>
```

#### 7.14.2.10 DiffusionDigitPutVL

```
using DiffusionDigitPutVL = pde::Diffusion<opt::PayoffDigitPut, pde::LocalVol>
```

#### 7.14.2.11 DiffusionDoubleDigitBS

```
using DiffusionDoubleDigitBS = pde::Diffusion<opt::PayoffDoubleDigit, pde::BSVol>
```

#### 7.14.2.12 DiffusionDoubleDigitVL

```
using DiffusionDoubleDigitVL = pde::Diffusion<opt::PayoffDoubleDigit, pde::LocalVol>
```

#### 7.14.2.13 DiffusionPutBS

```
using DiffusionPutBS = pde::Diffusion<opt::PayoffPut, pde::BSVol>
```

#### 7.14.2.14 DiffusionPutVL

```
using DiffusionPutVL = pde::Diffusion<opt::PayoffPut, pde::LocalVol>
```

#### 7.14.2.15 DiffusionStrangleBS

```
using DiffusionStrangleBS = pde::Diffusion<opt::PayoffStrangle, pde::BSVol>
```

#### 7.14.2.16 DiffusionStrangleVL

```
using DiffusionStrangleVL = pde::Diffusion<opt::PayoffStrangle, pde::LocalVol>
```

### 7.14.3 Documentation des fonctions

#### 7.14.3.1 \_\_declspec()

```
__declspec(dllexport)
```

Calcule le delta d'un call européen.

Calcule la grille du delta VL d'un butterfly.

Calcule la grille du prix VL d'un butterfly.

Calcule le delta VL d'un butterfly.

Calcule le prix VL d'un butterfly.

Calcule la grille du delta VL d'un strangle.

Calcule la grille du prix VL d'un strangle.

Calcule le delta VL d'un strangle.

Calcule le prix VL d'un strangle.

Calcule la grille du delta VL d'un bear spread.

Calcule la grille du prix VL d'un bear spread.

Calcule le delta VL d'un bear spread.

Calcule le prix VL d'un bear spread.

Calcule la grille du delta VL d'un bull spread.

Calcule la grille du prix VL d'un bull spread.

Calcule le delta VL d'un bull spread.

Calcule le prix VL d'un bull spread.

Calcule la grille du delta VL d'un double-digital.

Calcule la grille du prix VL d'un double-digital.

Calcule le delta VL d'un double-digital.

Calcule le prix VL d'un double-digital.

Calcule la grille du delta VL d'un put digital.

Calcule la grille du prix VL d'un put digital.

Calcule le delta VL d'un put digital.

Calcule le prix VL d'un put digital.

Calcule la grille du delta VL d'un call digital.

Calcule la grille du prix VL d'un call digital.

Calcule le delta VL d'un call digital.

Calcule le prix VL d'un call digital.

Calcule la grille du delta VL d'un put vanille.

Calcule la grille du prix VL d'un put vanille.

Calcule le delta VL d'un put vanille.

Calcule le prix VL d'un put vanille.

Calcule la grille du delta VL d'un call vanille.

Calcule la grille du prix VL d'un call vanille.

Calcule le delta VL d'un call vanille.

Calcule le prix VL d'un call vanille.

Calcule la volatilité locale.

Calcule la grille du delta BS d'un butterfly.

Calcule la grille du prix BS d'un butterfly.

Calcule le delta BS d'un butterfly.

Calcule le prix BS d'un butterfly.

Calcule la grille du delta BS d'un strangle.

Calcule la grille du prix BS d'un strangle.

Calcule le delta BS d'un strangle.

Calcule le prix BS d'un strangle.

Calcule la grille du delta BS d'un bear spread.

Calcule la grille du prix BS d'un bear spread.

Calcule le delta BS d'un bear spread.

Calcule le prix BS d'un bear spread.

Calcule la grille du delta BS d'un bull spread.

Calcule la grille du prix BS d'un bull spread.

Calcule le delta BS d'un bull spread.

Calcule le prix BS d'un bull spread.

Calcule la grille du delta BS d'un double-digital.

Calcule la grille du prix BS d'un double-digital.

Calcule le delta BS d'un double-digital.

Calcule le prix BS d'un double-digital.

Calcule la grille du delta BS d'un put digital.

Calcule la grille du prix BS d'un put digital.

Calcule le delta BS d'un put digital.

Calcule le prix BS d'un put digital.

Calcule la grille du delta BS d'un call digital.

Calcule la grille du prix BS d'un call digital.

Calcule le delta BS d'un call digital.

Calcule le prix BS d'un call digital.

Calcule la grille du delta BS d'un put vanille.

Calcule la grille du prix BS d'un put vanille.

Calcule le delta BS d'un put vanille.

Calcule le prix BS d'un put vanille.

Calcule la grille du delta BS d'un call vanille.

Calcule la grille du prix BS d'un call vanille.

Calcule le delta BS d'un call vanille.

Calcule le prix BS d'un call vanille.

Calcule le delta Repeated Richardson d'un put américain.

Calcule le prix Repeated Richardson d'un put américain.

Renvoie la matrice des bonds du put américain.

Renvoie la matrice des deltas du put américain.

Renvoie la matrice de valorisation du put américain.

Calcule le delta d'un put américain.

Calcule le prix d'un put américain.

Renvoie la matrice des bonds du call américain.

Renvoie la matrice des deltas du call américain.

Renvoie la matrice de valorisation du call américain.

Calcule le delta d'un call américain.

Calcule le prix d'un call américain.

Calcule le delta MC d'un put sur min.

Calcule le prix MC d'un put sur min.

Renvoie la matrice des bonds du put sur min.

Renvoie la matrice des deltas du put sur min.

Renvoie la matrice de valorisation du put sur min.

Calcule le delta d'un put sur min.

Calcule le prix d'un put sur min.

Calcule le delta MC d'un call sur max.

Calcule le prix MC d'un call sur max.

Renvoie la matrice des bonds du call sur max.

Renvoie la matrice des deltas du call sur max.

Renvoie la matrice de valorisation du call sur max.

Calcule le delta d'un call sur max.

Calcule le prix d'un call sur max.

Calcule le delta MC d'un put sur moyenne géométrique.

Calcule le prix MC d'un put sur moyenne géométrique.

Renvoie la matrice des bonds du put sur moyenne géométrique.

Renvoie la matrice des deltas du put sur moyenne géométrique.

Renvoie la matrice de valorisation du put sur moyenne géométrique.

Calcule le delta d'un put sur moyenne géométrique.

Calcule le prix d'un put sur moyenne géométrique.

Calcule le delta MC d'un call sur moyenne géométrique.

Calcule le prix MC d'un call sur moyenne géométrique.

Renvoie la matrice des bonds du call sur moyenne géométrique.

Renvoie la matrice des deltas du call sur moyenne géométrique.

Renvoie la matrice de valorisation du call sur moyenne géométrique.

Calcule le delta d'un call sur moyenne géométrique.

Calcule le prix d'un call sur moyenne géométrique.

Calcule le delta MC d'un put sur moyenne arithmétique.

Calcule le prix MC d'un put sur moyenne arithmétique.

Renvoie la matrice des bonds du put sur moyenne arithmétique.

Renvoie la matrice des deltas du put sur moyenne arithmétique.

Renvoie la matrice de valorisation du put sur moyenne arithmétique.

Calcule le delta d'un put sur moyenne arithmétique.

Calcule le prix d'un put sur moyenne arithmétique.

Calcule le delta MC d'un call sur moyenne arithmétique.

Calcule le prix MC d'un call sur moyenne arithmétique.

Renvoie la matrice des bonds du call sur moyenne arithmétique.

Renvoie la matrice des deltas du call sur moyenne arithmétique.

Renvoie la matrice de valorisation du call sur moyenne arithmétique.

Calcule le delta d'un call sur moyenne arithmétique.

Calcule le prix d'un call sur moyenne arithmétique.

Renvoie la matrice des bonds du butterfly.

Renvoie la matrice des deltas du butterfly.

Renvoie la matrice de valorisation du butterfly.

Calcule le delta d'une option butterfly.

Calcule le prix d'une option butterfly.

Renvoie la matrice des bonds du strangle.

Renvoie la matrice des deltas du strangle.

Renvoie la matrice de valorisation du strangle.

Calcule le delta d'une option strangle.

Calcule le prix d'une option strangle.

Renvoie la matrice des bonds du bear spread.

Renvoie la matrice des deltas du bear spread.

Renvoie la matrice de valorisation du bear spread.

Calcule le delta d'une option bear spread.

Calcule le prix d'une option bear spread.

Renvoie la matrice des bonds du bull spread.

Renvoie la matrice des deltas du bull spread.

Renvoie la matrice de valorisation du bull spread.

Calcule le delta d'une option bull spread.

Calcule le prix d'une option bull spread.

Renvoie la matrice des bonds du double-digit.

Renvoie la matrice des deltas du double-digit.

Renvoie la matrice de valorisation du double-digit.

Calcule le delta d'une option double-digital.

Calcule le prix d'une option double-digital.

Renvoie la matrice des bonds du put digital.

Renvoie la matrice des deltas du put digital.

Renvoie la matrice de valorisation du put digital.

Calcule le delta d'un put digital.

Calcule le prix d'un put digital.

Renvoie la matrice des bonds du call digital.

Renvoie la matrice des deltas du call digital.

Renvoie la matrice de valorisation du call digital.

Calcule le delta d'un call digital.

Calcule le prix d'un call digital.

Renvoie la matrice des bonds du put vanille.

Renvoie la matrice des deltas du put vanille.

Renvoie la matrice de valorisation du put vanille.

Calcule le delta d'un put européen.

Calcule le prix d'un put européen.

Renvoie la matrice des bonds du call vanille en VARIANT SAFEARRAY.

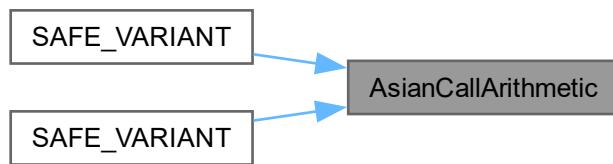
Renvoie la matrice des deltas du call vanille en VARIANT SAFEARRAY.

Renvoie la matrice 2D V[n][i] du call vanille en VARIANT SAFEARRAY.

### 7.14.3.2 AsianCallArithmetc()

```
double double double int double return AsianCallArithmetc (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffCall(K) ,
 crr::Arithmetc())
```

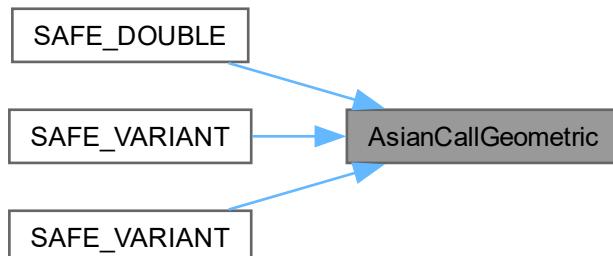
Voici le graphe des appelants de cette fonction :



### 7.14.3.3 AsianCallGeometric()

```
double double double int double return AsianCallGeometric (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffCall(K) ,
 crr::Geometric())
```

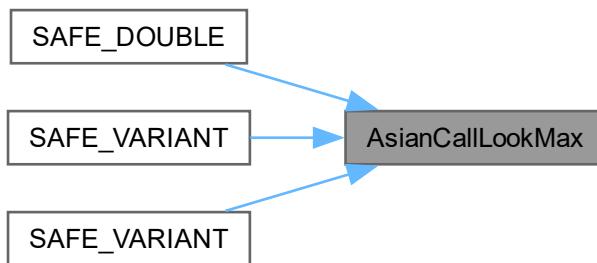
Voici le graphe des appelants de cette fonction :



#### 7.14.3.4 AsianCallLookMax()

```
double double double int double return AsianCallLookMax (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffCall(K) ,
 crr::LookMax())
```

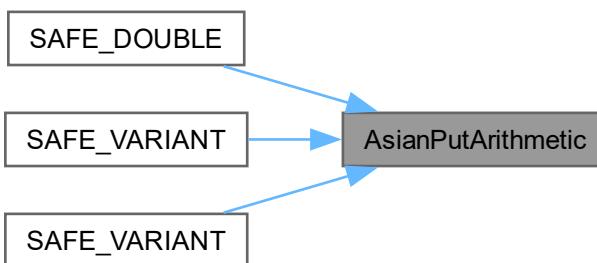
Voici le graphe des appelleurs de cette fonction :



#### 7.14.3.5 AsianPutArithmetic()

```
double double double int double return AsianPutArithmetic (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffPut(K) ,
 crr::Arithmetic())
```

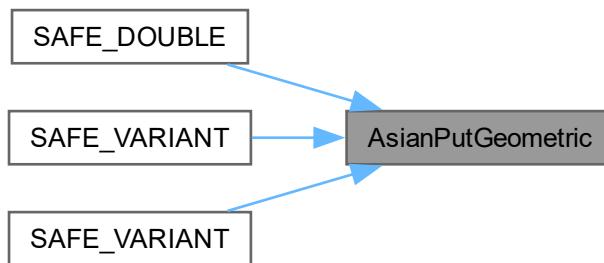
Voici le graphe des appelleurs de cette fonction :



#### 7.14.3.6 AsianPutGeometric()

```
double double double int double return AsianPutGeometric (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffPut(K) ,
 crr::Geometric())
```

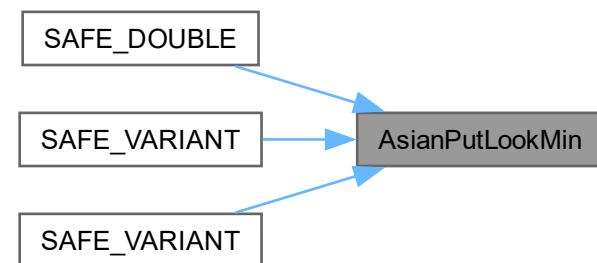
Voici le graphe des appels de cette fonction :



#### 7.14.3.7 AsianPutLookMin()

```
double double double int double return AsianPutLookMin (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffPut(K) ,
 crr::LookMin())
```

Voici le graphe des appels de cette fonction :



#### 7.14.3.8 `crr::American< opt::PayoffCall >()`

```
double double double int double return crr::American< opt::PayoffCall > (
 s0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffCall(K))
```

#### 7.14.3.9 `crr::American< opt::PayoffPut >()`

```
double double double int double return crr::American< opt::PayoffPut > (
 s0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffPut(K))
```

#### 7.14.3.10 `crr::European< opt::PayoffBear >()`

```
double double double int double double return crr::European< opt::PayoffBear > (
 s0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffBear(K1, K2))
```

#### 7.14.3.11 `crr::European< opt::PayoffBull >()`

```
double double double int double double return crr::European< opt::PayoffBull > (
 s0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffBull(K1, K2))
```

#### 7.14.3.12 `crr::European< opt::PayoffButterfly >()`

```
double double double int double double return crr::European< opt::PayoffButterfly > (
 s0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffButterfly(K1, K2))
```

**7.14.3.13 crr::European< opt::PayoffCall >()**

```
double double double int double return crr::European< opt::PayoffCall > (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffCall(K))
```

**7.14.3.14 crr::European< opt::PayoffDigitCall >()**

```
double double double int double return crr::European< opt::PayoffDigitCall > (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffDigitCall(K))
```

**7.14.3.15 crr::European< opt::PayoffDigitPut >()**

```
double double double int double return crr::European< opt::PayoffDigitPut > (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffDigitPut(K))
```

**7.14.3.16 crr::European< opt::PayoffDoubleDigit >()**

```
double double double int double double return crr::European< opt::PayoffDoubleDigit > (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffDoubleDigit(K1, K2))
```

**7.14.3.17 crr::European< opt::PayoffPut >()**

```
double double double int double return crr::European< opt::PayoffPut > (
 S0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffPut(K))
```

### 7.14.3.18 `crr::European< opt::PayoffStrangle >()`

```
double double double double int double double return crr::European< opt::PayoffStrangle > (
 s0 ,
 R ,
 sigma ,
 T ,
 N ,
 opt::PayoffStrangle(K1, K2))
```

### 7.14.3.19 `delta()`

```
return solver delta (
 t ,
 s)
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



### 7.14.3.20 `makeVariantFromArray()`

```
VARIANT makeVariantFromArray (
 SAFEARRAY * psa)
```

Emballe un SAFEARRAY\* de doubles dans un VARIANT.

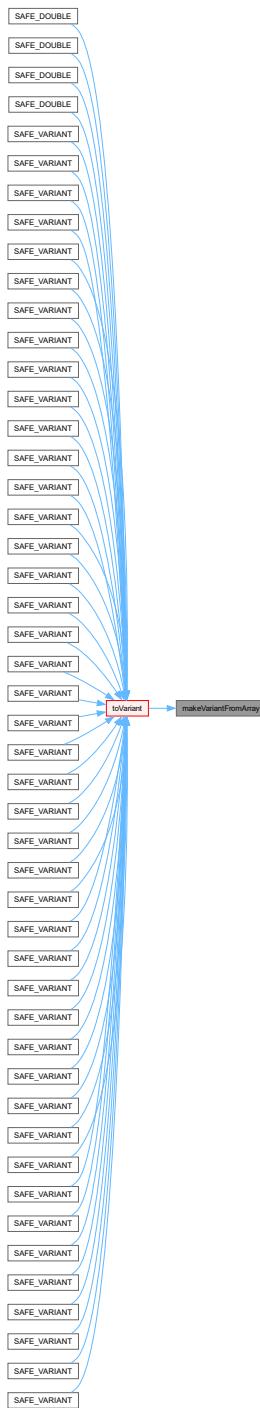
#### Paramètres

|                  |                                             |
|------------------|---------------------------------------------|
| <code>psa</code> | Le pointeur vers le SAFEARRAY à encapsuler. |
|------------------|---------------------------------------------|

**Renvoie**

Un VARIANT prêt à être renvoyé à VBA.

Voici le graphe des appels de cette fonction :



#### 7.14.3.21 reportError()

```
double reportError (
```

```
const char * msg,
const char * title)
```

Affiche une boîte d'erreur une seule fois.

#### Paramètres

|              |                                   |
|--------------|-----------------------------------|
| <i>msg</i>   | Le message d'erreur à afficher.   |
| <i>title</i> | Le titre de la boîte de dialogue. |

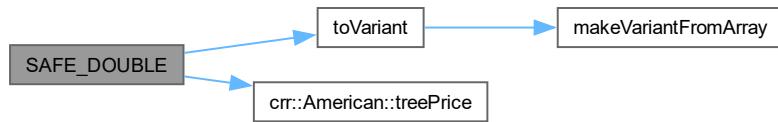
#### Renvoie

NaN pour signaler l'erreur au code appelant.

### 7.14.3.22 SAFE\_DOUBLE() [1/43]

```
SAFE_DOUBLE (
 DeltaAmCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::American< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).delta←
Zero();)
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.23 SAFE\_DOUBLE() [2/43]

```
SAFE_DOUBLE (
 DeltaAmPutRR ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).delta←
RR();)
```

### 7.14.3.24 SAFE\_DOUBLE() [3/43]

```
SAFE_DOUBLE (
 DeltaAritCallMC ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianCallArithmetic(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).delta←
MC();)
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.25 SAFE\_DOUBLE() [4/43]

```
SAFE_DOUBLE (
 DeltaAritPutMC ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).delta←
MC();)
```

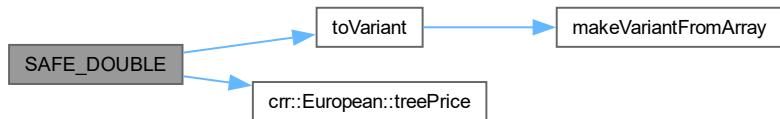
Voici le graphe d'appel pour cette fonction :



### 7.14.3.26 SAFE\_DOUBLE() [5/43]

```
SAFE_DOUBLE (
 DeltaBull ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 return crr::European< opt::PayoffBull >(S0, R, sigma, T, N, opt::PayoffBull(K1,
K2)).deltaZero();)
```

Voici le graphe d'appel pour cette fonction :



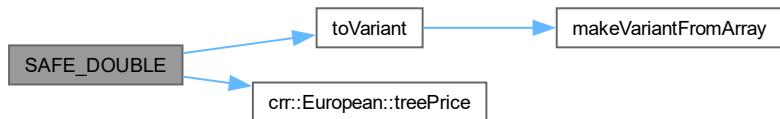
#### 7.14.3.27 `SAFE_DOUBLE()` [6/43]

```

SAFE_DOUBLE (
 DeltaDigitPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::European< opt::PayoffDigitPut >(S0, R, sigma, T, N, opt::PayoffDigitPut(K)).delta←
Zero();)

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.28 `SAFE_DOUBLE()` [7/43]

```

SAFE_DOUBLE (
 DeltaEuPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::European< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).delta←
Zero();)

```

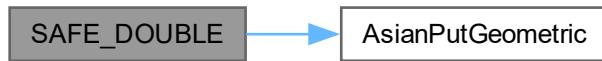
Voici le graphe d'appel pour cette fonction :



**7.14.3.29 SAFE\_DOUBLE() [8/43]**

```
SAFE_DOUBLE (
 DeltaGeomCallMC ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric()).delta←
MC();)
```

Voici le graphe d'appel pour cette fonction :

**7.14.3.30 SAFE\_DOUBLE() [9/43]**

```
SAFE_DOUBLE (
 DeltaGeomPutMC ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric()).delta←
MC();)
```

Voici le graphe d'appel pour cette fonction :

**7.14.3.31 SAFE\_DOUBLE() [10/43]**

```
SAFE_DOUBLE (
 DeltaMaxCallMC ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax()).delta←
MC();)
```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.32 SAFE\_DOUBLE() [11/43]

```

SAFE_DOUBLE (
 DeltaMinPutMC ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin()).delta←
MC();)

```

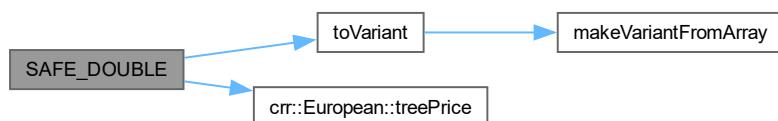
#### 7.14.3.33 SAFE\_DOUBLE() [12/43]

```

SAFE_DOUBLE (
 DeltaStrangle ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 return crr::European< opt::PayoffStrangle >(S0, R, sigma, T, N, opt::PayoffStrangle(K1,
K2)).deltaZero();)

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.34 SAFE\_DOUBLE() [13/43]

```

SAFE_DOUBLE (
 PriceAmPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).price();
)

```

### 7.14.3.35 SAFE\_DOUBLE() [14/43]

```
SAFE_DOUBLE (
 PriceAritCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianCallArithmetic(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetic()).price();
)
```

### 7.14.3.36 SAFE\_DOUBLE() [15/43]

```
SAFE_DOUBLE (
 PriceAritPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).price();
)
```

### 7.14.3.37 SAFE\_DOUBLE() [16/43]

```
SAFE_DOUBLE (
 PriceBear ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 return crr::European< opt::PayoffBear >(S0, R, sigma, T, N, opt::PayoffBear(K1,
K2)).price();)
```

### 7.14.3.38 SAFE\_DOUBLE() [17/43]

```
SAFE_DOUBLE (
 PriceBearBS ,
 (double t, double S, double sigma, double T, double R, double K1, double K2,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionBearBS eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::BSVol(sigma));pde::CNMethod<
DiffusionBearBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.39 SAFE\_DOUBLE() [18/43]

```
SAFE_DOUBLE (
 PriceBearVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K1,
double K2, double Smin, double Smax, int imax, int jmax) ,
 { DiffusionBearVL eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionBearVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.~
v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.40 SAFE\_DOUBLE() [19/43]

```
SAFE_DOUBLE (
 PriceBullBS ,
 (double t, double S, double sigma, double T, double R, double K1, double K2,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionBullBS eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::BSVol(sigma));pde::CNMethod<
DiffusionBullBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.41 SAFE\_DOUBLE() [20/43]

```
SAFE_DOUBLE (
 PriceBullVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K1,
double K2, double Smin, double Smax, int imax, int jmax) ,
 { DiffusionBullVL eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionBullVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.~
v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.42 SAFE\_DOUBLE() [21/43]

```

SAFE_DOUBLE (
 PriceButterfly ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 return crr::European< opt::PayoffButterfly >(S0, R, sigma, T, N, opt::PayoffButterfly(K1,
K2)).price();)

```

#### 7.14.3.43 SAFE\_DOUBLE() [22/43]

```

SAFE_DOUBLE (
 PriceButterflyBS ,
 (double t, double S, double sigma, double T, double R, double K1, double K2,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionButterflyBS eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::BSVol(sigma));pde:
DiffusionButterflyBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.44 SAFE\_DOUBLE() [23/43]

```

SAFE_DOUBLE (
 PriceButterflyVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K1,
double K2, double Smin, double Smax, int imax, int jmax) ,
 { DiffusionButterflyVL eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionButterflyVL > solver(eq, imax, jmax);solver.SolvePDE();return
solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.45 SAFE\_DOUBLE() [24/43]

```

SAFE_DOUBLE (
 PriceDD ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 return crr::European< opt::PayoffDoubleDigit >(S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1,
K2)).price();)

```

#### 7.14.3.46 SAFE\_DOUBLE() [25/43]

```

SAFE_DOUBLE (
 PriceDDBS ,
 (double t, double S, double sigma, double T, double R, double K1, double K2,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionDoubleDigitBS eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2),
pde::BSVol(sigma));pde::CNMethod< DiffusionDoubleDigitBS > solver(eq, imax, jmax);solver.SolvePDE();return
solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.47 SAFE\_DOUBLE() [26/43]

```

SAFE_DOUBLE (
 PriceDDVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K1,
double K2, double Smin, double Smax, int imax, int jmax) ,
 { DiffusionDoubleDigitVL eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2),

```

```
pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDoubleDigitVL > solver(eq, imax, jmax);solver.SolvePDE();r
solver.v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.48 SAFE\_DOUBLE() [27/43]

```
SAFE_DOUBLE (
 PriceDigitCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::European< opt::PayoffDigitCall >(S0, R, sigma, T, N, opt::PayoffDigitCall(K)).price(
)
```

#### 7.14.3.49 SAFE\_DOUBLE() [28/43]

```
SAFE_DOUBLE (
 PriceDigitCallBS ,
 (double t, double S, double sigma, double T, double R, double K, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionDigitCallBS eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::BSVol(sigma));pde::CNMe
DiffusionDigitCallBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.50 SAFE\_DOUBLE() [29/43]

```
SAFE_DOUBLE (
 PriceDigitCallVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K,
double Smin, double Smax, int imax, int jmax) ,
```

```

 { DiffusionDigitCallVL eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionDigitCallVL > solver(eq, imax, jmax);solver.SolvePDE();return
solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.51 SAFE\_DOUBLE() [30/43]

```

SAFE_DOUBLE (
 PriceDigitPutBS ,
 (double t, double S, double sigma, double T, double R, double K, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionDigitPutBS eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::BSVol(sigma));pde::CNMethod<
DiffusionDigitPutBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.52 SAFE\_DOUBLE() [31/43]

```

SAFE_DOUBLE (
 PriceDigitPutVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionDigitPutVL eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionDigitPutVL > solver(eq, imax, jmax);solver.SolvePDE();return
solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.53 SAFE\_DOUBLE() [32/43]

```

SAFE_DOUBLE (
 PriceEuCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return crr::European< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).price();
)

```

#### 7.14.3.54 SAFE\_DOUBLE() [33/43]

```

SAFE_DOUBLE (
 PriceEuCallBS ,
 (double t, double S, double sigma, double T, double R, double K, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionCallBS eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::BSVol(sigma));pde::CNMethod<
DiffusionCallBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.55 SAFE\_DOUBLE() [34/43]

```

SAFE_DOUBLE (
 PriceEuCallVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionCallVL eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionCallVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.56 SAFE\_DOUBLE() [35/43]

```

SAFE_DOUBLE (
 PriceEuPutBS ,
 (double t, double S, double sigma, double T, double R, double K, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionPutBS eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::BSVol(sigma));pde::CNMethod<
DiffusionPutBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



#### 7.14.3.57 SAFE\_DOUBLE() [36/43]

```

SAFE_DOUBLE (
 PriceEuPutVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K,
double Smin, double Smax, int imax, int jmax) ,
 { DiffusionPutVL eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionPutVL > solver(eq, imax, jmax);solver.SolvePDE();return solver.←
v(t, S);})

```

Voici le graphe d'appel pour cette fonction :



**7.14.3.58 SAFE\_DOUBLE() [37/43]**

```
SAFE_DOUBLE (
 PriceGeomCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric()).price();
)
```

**7.14.3.59 SAFE\_DOUBLE() [38/43]**

```
SAFE_DOUBLE (
 PriceGeomPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric()).price();
)
```

**7.14.3.60 SAFE\_DOUBLE() [39/43]**

```
SAFE_DOUBLE (
 PriceMaxCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax()).price();
)
```

**7.14.3.61 SAFE\_DOUBLE() [40/43]**

```
SAFE_DOUBLE (
 PriceMinPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 return AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin()).price();
)
```

**7.14.3.62 SAFE\_DOUBLE() [41/43]**

```
SAFE_DOUBLE (
 PriceStrangleBS ,
 (double t, double S, double sigma, double T, double R, double K1, double K2,
 double Smin, double Smax, int imax, int jmax) ,
 { DiffusionStrangleBS eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::BSVol(sigma));pde::
 DiffusionStrangleBS > solver(eq, imax, jmax);solver.SolvePDE();return solver.v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.63 SAFE\_DOUBLE() [42/43]

```
SAFE_DOUBLE (
 PriceStrangleVL ,
 (double t, double S, double alfa, double beta, double T, double R, double K1,
double K2, double Smin, double Smax, int imax, int jmax) ,
 { DiffusionStrangleVL eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionStrangleVL > solver(eq, imax, jmax);solver.SolvePDE();return
solver.v(t, S);})
```

Voici le graphe d'appel pour cette fonction :



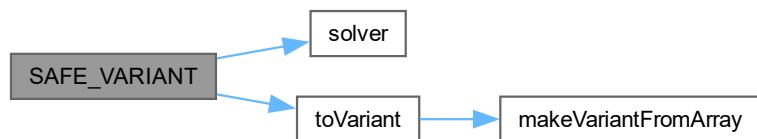
### 7.14.3.64 SAFE\_DOUBLE() [43/43]

```
SAFE_DOUBLE (
 Vol ,
 (double t, double S, double alfa, double beta) ,
 return pde::LocalVol(alfa, beta)(t, S);)
```

### 7.14.3.65 SAFE\_VARIANT() [1/47]

```
SAFE_VARIANT (
 GridPriceBearBS ,
 (double sigma, double T, double R, double K1, double K2, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionBearBS eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::BSVol(sigma));pde::CNMethod<
DiffusionBearBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.<-
val);})
```

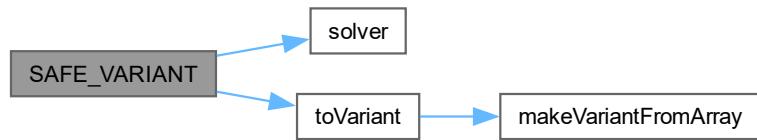
Voici le graphe d'appel pour cette fonction :



### 7.14.3.66 SAFE\_VARIANT() [2/47]

```
SAFE_VARIANT (
 GridPriceBearVL ,
 (double alfa, double beta, double T, double R, double K1, double K2, double Smin,
 double Smax, int imax, int jmax) ,
 { DiffusionBearVL eq(T, Smin, Smax, R, opt::PayoffBear(K1, K2), pde::LocalVol(alfa,
 beta));pde::CNMethod< DiffusionBearVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.~
 grid();return toVariant(G.val);})
```

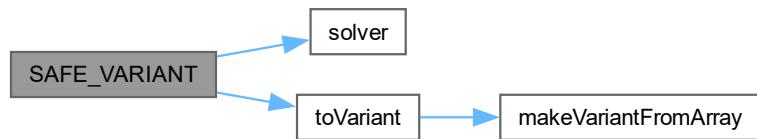
Voici le graphe d'appel pour cette fonction :



### 7.14.3.67 SAFE\_VARIANT() [3/47]

```
SAFE_VARIANT (
 GridPriceBullBS ,
 (double sigma, double T, double R, double K1, double K2, double Smin, double
 Smax, int imax, int jmax) ,
 { DiffusionBullBS eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::BSVol(sigma));pde::CNMethod<
 DiffusionBullBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.~
 val);})
```

Voici le graphe d'appel pour cette fonction :

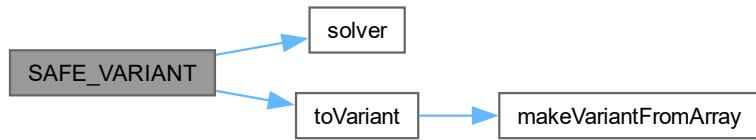


### 7.14.3.68 SAFE\_VARIANT() [4/47]

```
SAFE_VARIANT (
 GridPriceBullVL ,
 (double alfa, double beta, double T, double R, double K1, double K2, double Smin,
```

```
double Smax, int imax, int jmax) ,
{ DiffusionBullVL eq(T, Smin, Smax, R, opt::PayoffBull(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionBullVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.←
grid();return toVariant(G.val);})
```

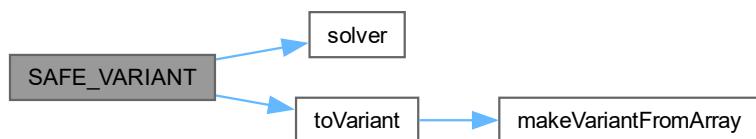
Voici le graphe d'appel pour cette fonction :



#### 7.14.3.69 SAFE\_VARIANT() [5/47]

```
SAFE_VARIANT (
 GridPriceButterflyBS ,
 (double sigma, double T, double R, double K1, double K2, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionButterflyBS eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::BSVol(sigma));pde:
DiffusionButterflyBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return
toVariant(G.val);})
```

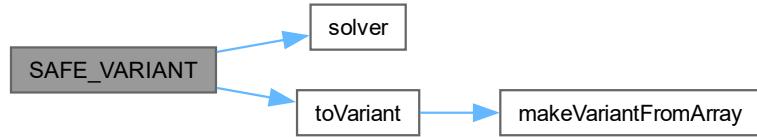
Voici le graphe d'appel pour cette fonction :



#### 7.14.3.70 SAFE\_VARIANT() [6/47]

```
SAFE_VARIANT (
 GridPriceButterflyVL ,
 (double alfa, double beta, double T, double R, double K1, double K2, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionButterflyVL eq(T, Smin, Smax, R, opt::PayoffButterfly(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionButterflyVL > solver(eq, imax, jmax);solver.SolvePDE();auto
G=solver.grid();return toVariant(G.val);})
```

Voici le graphe d'appel pour cette fonction :



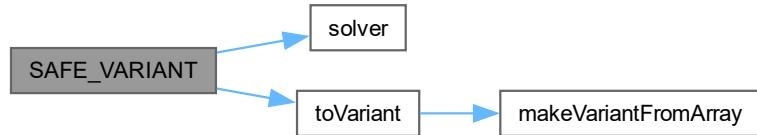
#### 7.14.3.71 `SAFE_VARIANT()` [7/47]

```

SAFE_VARIANT (
 GridPriceDDBS ,
 (double sigma, double T, double R, double K1, double K2, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionDoubleDigitBS eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2),
pde::BSVol(sigma));pde::CNMethod< DiffusionDoubleDigitBS > solver(eq, imax, jmax);solver.SolvePDE();auto
G=solver.grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



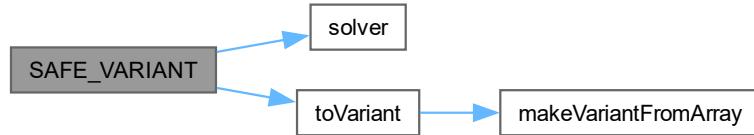
#### 7.14.3.72 `SAFE_VARIANT()` [8/47]

```

SAFE_VARIANT (
 GridPriceDDVL ,
 (double alfa, double beta, double T, double R, double K1, double K2, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionDoubleDigitVL eq(T, Smin, Smax, R, opt::PayoffDoubleDigit(K1, K2),
pde::LocalVol(alfa, beta));pde::CNMethod< DiffusionDoubleDigitVL > solver(eq, imax, jmax);solver.SolvePDE();auto
G=solver.grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



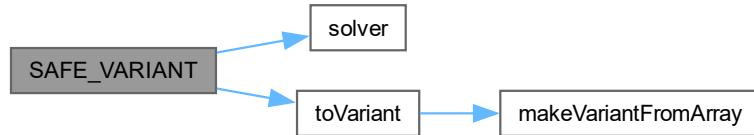
#### 7.14.3.73 `SAFE_VARIANT()` [9/47]

```

SAFE_VARIANT (
 GridPriceDigitCallBS ,
 (double sigma, double T, double R, double K, double Smin, double Smax, int imax,
int jmax) ,
 { DiffusionDigitCallBS eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::BSVol(sigma));pde::CNMe
DiffusionDigitCallBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return
toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



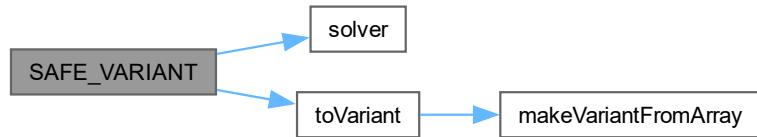
#### 7.14.3.74 `SAFE_VARIANT()` [10/47]

```

SAFE_VARIANT (
 GridPriceDigitCallVL ,
 (double alfa, double beta, double T, double R, double K, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionDigitCallVL eq(T, Smin, Smax, R, opt::PayoffDigitCall(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionDigitCallVL > solver(eq, imax, jmax);solver.SolvePDE();auto
G=solver.grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



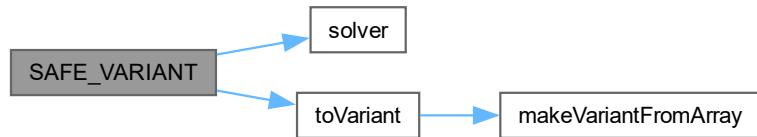
#### 7.14.3.75 `SAFE_VARIANT()` [11/47]

```

SAFE_VARIANT (
 GridPriceDigitPutBS ,
 (double sigma, double T, double R, double K, double Smin, double Smax, int imax,
int jmax) ,
 { DiffusionDigitPutBS eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::BSVol(sigma));pde::CNMethod
DiffusionDigitPutBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return
toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



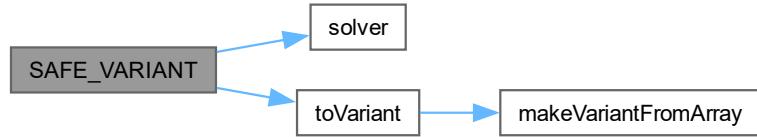
#### 7.14.3.76 `SAFE_VARIANT()` [12/47]

```

SAFE_VARIANT (
 GridPriceDigitPutVL ,
 (double alfa, double beta, double T, double R, double K, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionDigitPutVL eq(T, Smin, Smax, R, opt::PayoffDigitPut(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionDigitPutVL > solver(eq, imax, jmax);solver.SolvePDE();auto
G=solver.grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



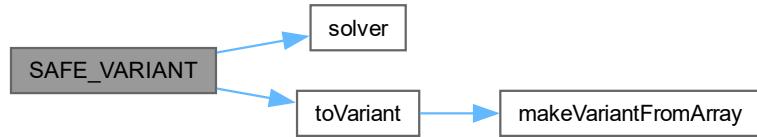
#### 7.14.3.77 `SAFE_VARIANT()` [13/47]

```

SAFE_VARIANT (
 GridPriceEuCallBS ,
 (double sigma, double T, double R, double K, double Smin, double Smax, int imax,
int jmax) ,
 { DiffusionCallBS eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::BSVol(sigma));pde::CNMethod<
DiffusionCallBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.<-
val);})

```

Voici le graphe d'appel pour cette fonction :



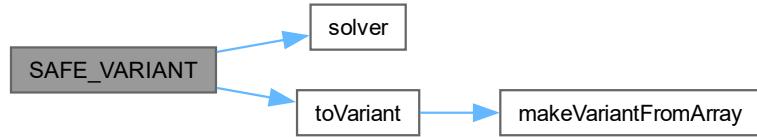
#### 7.14.3.78 `SAFE_VARIANT()` [14/47]

```

SAFE_VARIANT (
 GridPriceEuCallVL ,
 (double alfa, double beta, double T, double R, double K, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionCallVL eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionCallVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.<-
grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



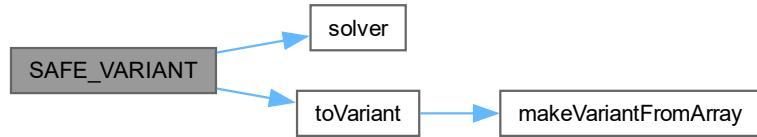
#### 7.14.3.79 SAFE\_VARIANT() [15/47]

```

SAFE_VARIANT (
 GridPriceEuPutBS ,
 (double sigma, double T, double R, double K, double Smin, double Smax, int imax,
int jmax) ,
 { DiffusionPutBS eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::BSVol(sigma));pde::CNMethod<
DiffusionPutBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return toVariant(G.<-
val);})

```

Voici le graphe d'appel pour cette fonction :



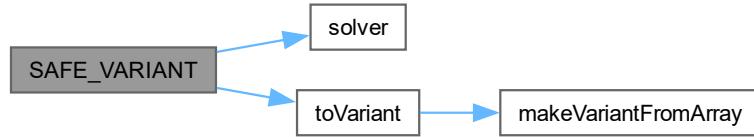
#### 7.14.3.80 SAFE\_VARIANT() [16/47]

```

SAFE_VARIANT (
 GridPriceEuPutVL ,
 (double alfa, double beta, double T, double R, double K, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionPutVL eq(T, Smin, Smax, R, opt::PayoffPut(K), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionPutVL > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.<-
grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



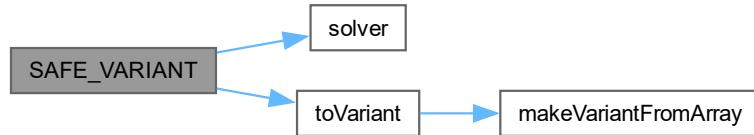
#### 7.14.3.81 `SAFE_VARIANT()` [17/47]

```

SAFE_VARIANT (
 GridPriceStrangleBS ,
 (double sigma, double T, double R, double K1, double K2, double Smin, double
Smax, int imax, int jmax) ,
 { DiffusionStrangleBS eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::BSVol(sigma));pde::C
DiffusionStrangleBS > solver(eq, imax, jmax);solver.SolvePDE();auto G=solver.grid();return
toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



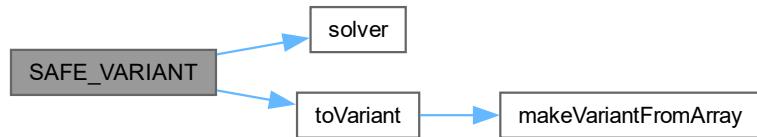
#### 7.14.3.82 `SAFE_VARIANT()` [18/47]

```

SAFE_VARIANT (
 GridPriceStrangleVL ,
 (double alfa, double beta, double T, double R, double K1, double K2, double Smin,
double Smax, int imax, int jmax) ,
 { DiffusionStrangleVL eq(T, Smin, Smax, R, opt::PayoffStrangle(K1, K2), pde::LocalVol(alfa,
beta));pde::CNMethod< DiffusionStrangleVL > solver(eq, imax, jmax);solver.SolvePDE();auto
G=solver.grid();return toVariant(G.val);})

```

Voici le graphe d'appel pour cette fonction :



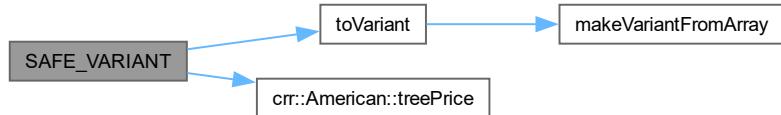
#### 7.14.3.83 SAFE\_VARIANT() [19/47]

```

SAFE_VARIANT (
 TreeAmPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).tree←
Price();return toVariant(V);})

```

Voici le graphe d'appel pour cette fonction :



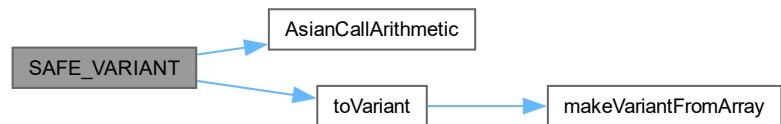
#### 7.14.3.84 SAFE\_VARIANT() [20/47]

```

SAFE_VARIANT (
 TreeAritCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=AsianCallArithmetric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetric()).tree←
Price();return toVariant(V);})

```

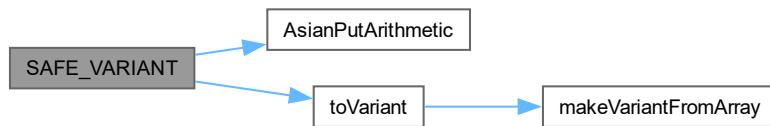
Voici le graphe d'appel pour cette fonction :



### 7.14.3.85 SAFE\_VARIANT() [21/47]

```
SAFE_VARIANT (
 TreeAritPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=AsianPutArithmetic(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetic()).treePrice(); return toVariant(V); })
```

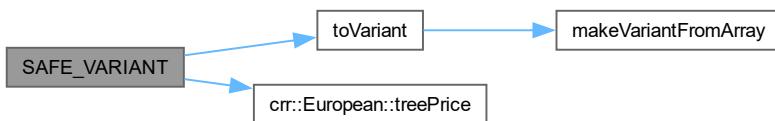
Voici le graphe d'appel pour cette fonction :



### 7.14.3.86 SAFE\_VARIANT() [22/47]

```
SAFE_VARIANT (
 TreeBear ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto V=crr::European< opt::PayoffBear >(S0, R, sigma, T, N, opt::PayoffBear(K1, K2)).treePrice(); return toVariant(V); })
```

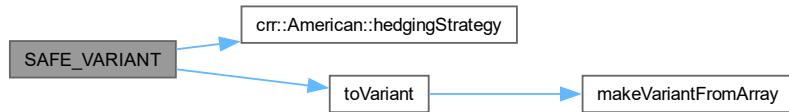
Voici le graphe d'appel pour cette fonction :



### 7.14.3.87 SAFE\_VARIANT() [23/47]

```
SAFE_VARIANT (
 TreeBondAmPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=crr::American< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).hedgingStrategy(); return toVariant(H.bond); })
```

Voici le graphe d'appel pour cette fonction :



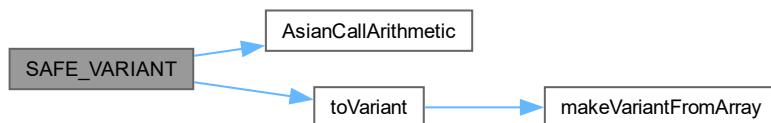
#### 7.14.3.88 SAFE\_VARIANT() [24/47]

```

SAFE_VARIANT (
 TreeBondAritCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=AsianCallArithmetric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Arithmetric()).hedging←
 Strategy(); return toVariant(H.bond); })

```

Voici le graphe d'appel pour cette fonction :



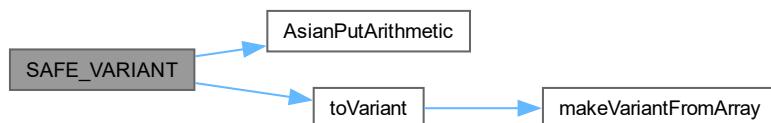
#### 7.14.3.89 SAFE\_VARIANT() [25/47]

```

SAFE_VARIANT (
 TreeBondAritPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=AsianPutArithmetric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Arithmetric()).hedging←
 Strategy(); return toVariant(H.bond); })

```

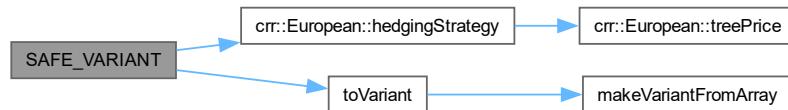
Voici le graphe d'appel pour cette fonction :



### 7.14.3.90 SAFE\_VARIANT() [26/47]

```
SAFE_VARIANT (
 TreeBondBear ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto H=crr::European< opt::PayoffBear >(S0, R, sigma, T, N, opt::PayoffBear(K1,
K2)).hedgingStrategy();return toVariant(H.bond);})
```

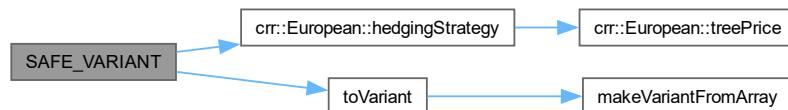
Voici le graphe d'appel pour cette fonction :



### 7.14.3.91 SAFE\_VARIANT() [27/47]

```
SAFE_VARIANT (
 TreeBondButterfly ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto H=crr::European< opt::PayoffButterfly >(S0, R, sigma, T, N, opt::PayoffButterfly(K1,
K2)).hedgingStrategy();return toVariant(H.bond);})
```

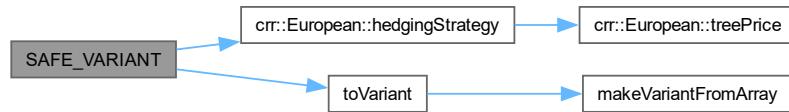
Voici le graphe d'appel pour cette fonction :



### 7.14.3.92 SAFE\_VARIANT() [28/47]

```
SAFE_VARIANT (
 TreeBondDD ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto H=crr::European< opt::PayoffDoubleDigit >(S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1,
K2)).hedgingStrategy();return toVariant(H.bond);})
```

Voici le graphe d'appel pour cette fonction :



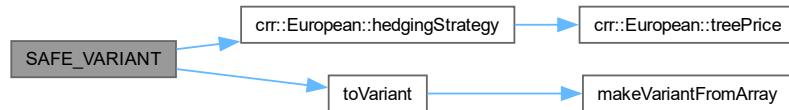
#### 7.14.3.93 SAFE\_VARIANT() [29/47]

```

SAFE_VARIANT (
 TreeBondDigitCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=crr::European< opt::PayoffDigitCall >(S0, R, sigma, T, N, opt::PayoffDigitCall(K)).hedg
Strategy(); return toVariant(H.bond); })

```

Voici le graphe d'appel pour cette fonction :



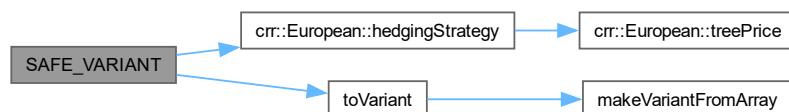
#### 7.14.3.94 SAFE\_VARIANT() [30/47]

```

SAFE_VARIANT (
 TreeBondEuCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=crr::European< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).hedging←
Strategy(); return toVariant(H.bond); })

```

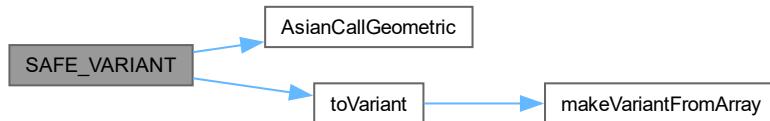
Voici le graphe d'appel pour cette fonction :



### 7.14.3.95 SAFE\_VARIANT() [31/47]

```
SAFE_VARIANT (
 TreeBondGeomCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric()).hedging←
Strategy(); return toVariant(H.bond); })
```

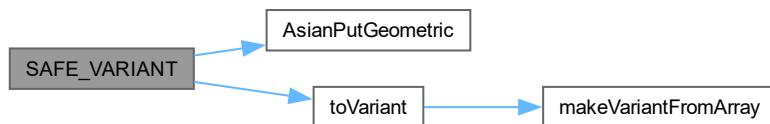
Voici le graphe d'appel pour cette fonction :



### 7.14.3.96 SAFE\_VARIANT() [32/47]

```
SAFE_VARIANT (
 TreeBondGeomPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric()).hedging←
Strategy(); return toVariant(H.bond); })
```

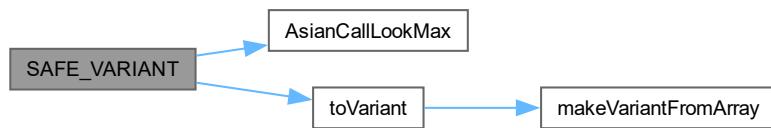
Voici le graphe d'appel pour cette fonction :



### 7.14.3.97 SAFE\_VARIANT() [33/47]

```
SAFE_VARIANT (
 TreeBondMaxCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax()).hedging←
Strategy(); return toVariant(H.bond); })
```

Voici le graphe d'appel pour cette fonction :



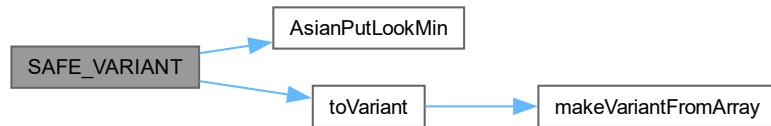
#### 7.14.3.98 SAFE\_VARIANT() [34/47]

```

SAFE_VARIANT (
 TreeBondMinPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin()).hedging<-
Strategy(); return toVariant(H.bond); })

```

Voici le graphe d'appel pour cette fonction :



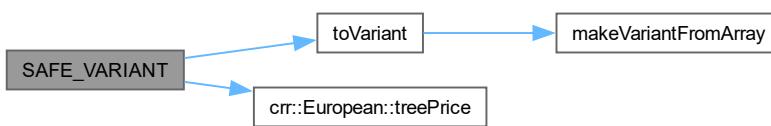
#### 7.14.3.99 SAFE\_VARIANT() [35/47]

```

SAFE_VARIANT (
 TreeButterfly ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto V=crr::European< opt::PayoffButterfly >(S0, R, sigma, T, N, opt::PayoffButterfly(K1,
K2)).treePrice(); return toVariant(V); })

```

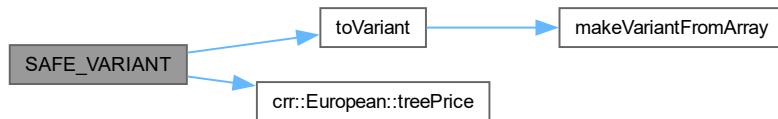
Voici le graphe d'appel pour cette fonction :



### 7.14.3.100 SAFE\_VARIANT() [36/47]

```
SAFE_VARIANT (
 TreeDD ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto V=crr::European< opt::PayoffDoubleDigit >(S0, R, sigma, T, N, opt::PayoffDoubleDigit(K1,
K2)).treePrice();return toVariant(V); })
```

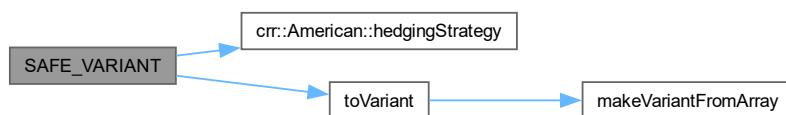
Voici le graphe d'appel pour cette fonction :



### 7.14.3.101 SAFE\_VARIANT() [37/47]

```
SAFE_VARIANT (
 TreeDeltaAmCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=crr::American< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).hedgingStrategy();
return toVariant(H.delta); })
```

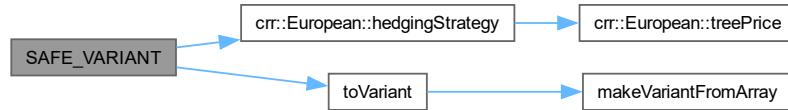
Voici le graphe d'appel pour cette fonction :



### 7.14.3.102 SAFE\_VARIANT() [38/47]

```
SAFE_VARIANT (
 TreeDeltaBull ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto H=crr::European< opt::PayoffBull >(S0, R, sigma, T, N, opt::PayoffBull(K1,
K2)).hedgingStrategy();return toVariant(H.delta); })
```

Voici le graphe d'appel pour cette fonction :



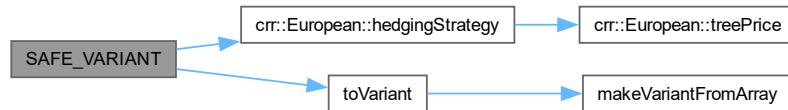
#### 7.14.3.103 SAFE\_VARIANT() [39/47]

```

SAFE_VARIANT (
 TreeDeltaDigitPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=crr::European< opt::PayoffDigitPut >(S0, R, sigma, T, N, opt::PayoffDigitPut(K)).hedgingStrategy(); return toVariant(H.delta); })

```

Voici le graphe d'appel pour cette fonction :



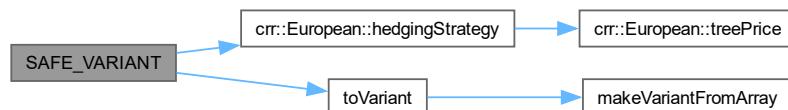
#### 7.14.3.104 SAFE\_VARIANT() [40/47]

```

SAFE_VARIANT (
 TreeDeltaEuPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto H=crr::European< opt::PayoffPut >(S0, R, sigma, T, N, opt::PayoffPut(K)).hedgingStrategy(); return toVariant(H.delta); })

```

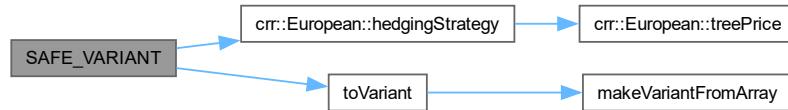
Voici le graphe d'appel pour cette fonction :



### 7.14.3.105 SAFE\_VARIANT() [41/47]

```
SAFE_VARIANT (
 TreeDeltaStrangle ,
 (double S0, double R, double sigma, double T, int N, double K1, double K2) ,
 { auto H=crr::European< opt::PayoffStrangle >(S0, R, sigma, T, N, opt::PayoffStrangle(K1,
K2)).hedgingStrategy();return toVariant(H.delta);})
```

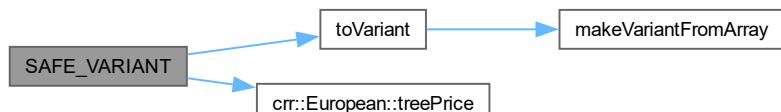
Voici le graphe d'appel pour cette fonction :



### 7.14.3.106 SAFE\_VARIANT() [42/47]

```
SAFE_VARIANT (
 TreeDigitCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=crr::European< opt::PayoffDigitCall >(S0, R, sigma, T, N, opt::PayoffDigitCall(K)).tree
Price();return toVariant(V);})
```

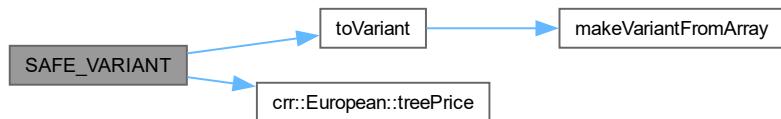
Voici le graphe d'appel pour cette fonction :



### 7.14.3.107 SAFE\_VARIANT() [43/47]

```
SAFE_VARIANT (
 TreeEuCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=crr::European< opt::PayoffCall >(S0, R, sigma, T, N, opt::PayoffCall(K)).tree←
Price();return toVariant(V);})
```

Voici le graphe d'appel pour cette fonction :



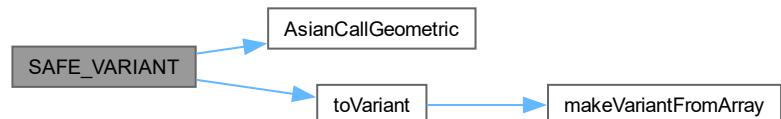
#### 7.14.3.108 SAFE\_VARIANT() [44/47]

```

SAFE_VARIANT (
 TreeGeomCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=AsianCallGeometric(S0, R, sigma, T, N, opt::PayoffCall(K), crr::Geometric()).tree←
Price();return toVariant(V); }
)

```

Voici le graphe d'appel pour cette fonction :



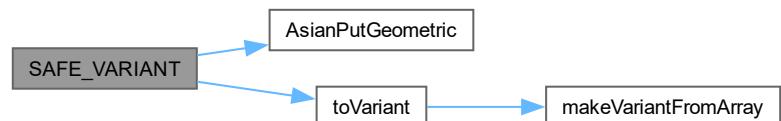
#### 7.14.3.109 SAFE\_VARIANT() [45/47]

```

SAFE_VARIANT (
 TreeGeomPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=AsianPutGeometric(S0, R, sigma, T, N, opt::PayoffPut(K), crr::Geometric()).tree←
Price();return toVariant(V); }
)

```

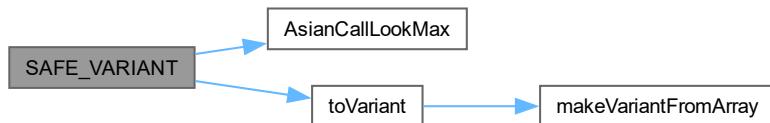
Voici le graphe d'appel pour cette fonction :



### 7.14.3.110 SAFE\_VARIANT() [46/47]

```
SAFE_VARIANT (
 TreeMaxCall ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=AsianCallLookMax(S0, R, sigma, T, N, opt::PayoffCall(K), crr::LookMax()).tree←
Price();return toVariant(V); })
```

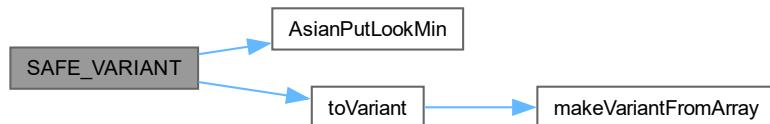
Voici le graphe d'appel pour cette fonction :



### 7.14.3.111 SAFE\_VARIANT() [47/47]

```
SAFE_VARIANT (
 TreeMinPut ,
 (double S0, double R, double sigma, double T, int N, double K) ,
 { auto V=AsianPutLookMin(S0, R, sigma, T, N, opt::PayoffPut(K), crr::LookMin()).tree←
Price();return toVariant(V); })
```

Voici le graphe d'appel pour cette fonction :



### 7.14.3.112 SolvePDE()

```
solver SolvePDE ()
```

Voici le graphe d'appel pour cette fonction :



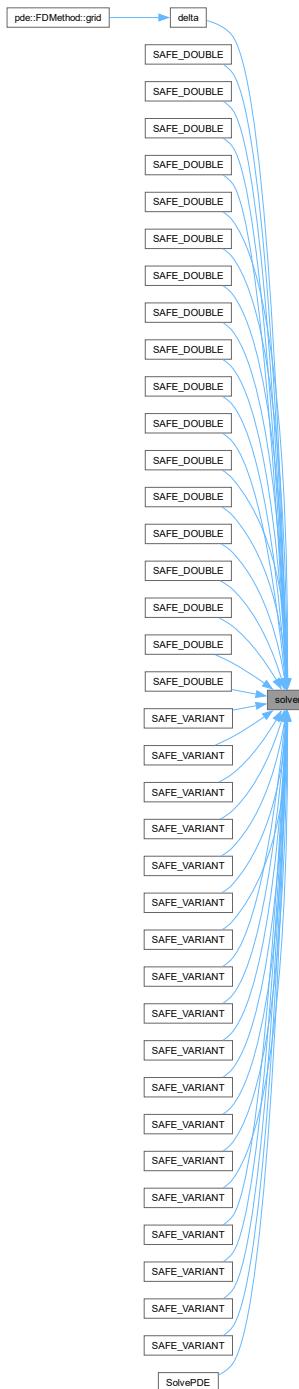
### 7.14.3.113 solver()

```
double double double double double double double double int int pde::CNMethod< DiffusionButterflyVL
> solver (
 eq ,
 imax ,
 jmax)
```

**Valeur initiale :**

```
{
 DiffusionCallBS eq(T, Smin, Smax, R, opt::PayoffCall(K), pde::BSVol(sigma))
```

Voici le graphe des appelants de cette fonction :



#### 7.14.3.114 toVariant() [1/4]

```
template<typename Mtx>
VARIANT toVariant (
 const Mtx & matrix)
```

Convertit un conteneur matriciel en VARIANT COM pour l'interopérabilité.

**Paramètres du template**

|            |                                      |
|------------|--------------------------------------|
| <i>Mtx</i> | Type du conteneur à deux dimensions. |
|------------|--------------------------------------|

**Paramètres**

|               |                                              |
|---------------|----------------------------------------------|
| <i>matrix</i> | Référence constante à la matrice de valeurs. |
|---------------|----------------------------------------------|

**Renvoie**

VARIANT contenant un SAFEARRAY bidimensionnel de doubles initialisé avec les éléments de la matrice.

Voici le graphe d'appel pour cette fonction :

**7.14.3.115 toVariant() [2/4]**

```
return toVariant (
 G. del)
```

**7.14.3.116 toVariant() [3/4]**

```
double double double double int double return toVariant (
 H. delta)
```

**Valeur initiale :**

```
{
 auto H = crr::European<opt::PayoffPut>(S0, R, sigma, T, N, opt::PayoffPut(K)).hedgingStrategy()
```

**7.14.3.117 toVariant() [4/4]**

```
double double double double int double return toVariant (
 V)
```

**Valeur initiale :**

```
{
 auto V = crr::European<opt::PayoffDigitPut>(S0, R, sigma, T, N, opt::PayoffDigitPut(K)).treePrice()
```

## 7.14.4 Documentation des variables

### 7.14.4.1 alfa

```
double double alfa
```

### 7.14.4.2 beta

```
double beta
```

### 7.14.4.3 G

```
auto G = solver.grid()
```

### 7.14.4.4 imax

```
double double double double double double double int imax
```

### 7.14.4.5 jmax

```
double double double double double double double int int jmax
```

### 7.14.4.6 K

```
double double double double K
```

### 7.14.4.7 K1

```
double double double double K1
```

### 7.14.4.8 K2

```
double double double double double K2
```

### 7.14.4.9 N

```
double double double int N
```

### 7.14.4.10 R

```
double double double R
```

**7.14.4.11 S**

```
double S
```

**7.14.4.12 S0**

```
double S0
```

**7.14.4.13 sigma**

```
double double sigma
```

**7.14.4.14 Smax**

```
double double double double double double Smax
```

**7.14.4.15 Smin**

```
double double double double double double Smin
```

**7.14.4.16 T**

```
double double T
```

**7.14.4.17 t**

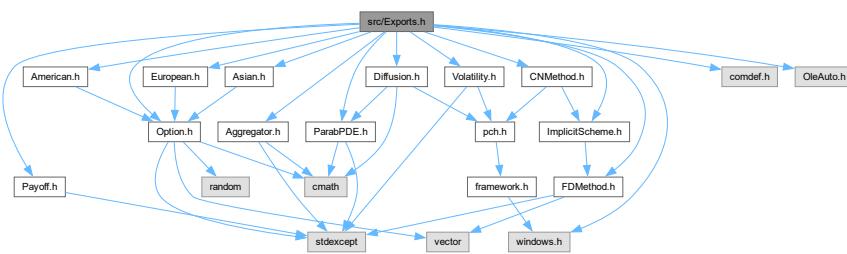
```
double t
```

**7.15 Référence du fichier src/Exports.h**

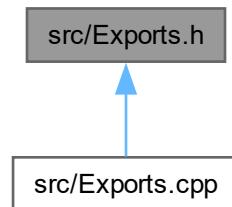
```
#include "Payoff.h"
#include "Option.h"
#include "European.h"
#include "Aggregator.h"
#include "Asian.h"
#include "American.h"
#include "ParabPDE.h"
#include "Volatility.h"
#include "Diffusion.h"
#include "FDMETHOD.h"
#include "ImplicitScheme.h"
#include "CNMethod.h"
#include <windows.h>
#include <comdef.h>
```

```
#include <OleAuto.h>
```

Graphe des dépendances par inclusion de Exports.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Fonctions

- **`__declspec(dllexport) double __stdcall PriceEuCall(double S0`**

*Calcule le prix d'un call européen.*

## Variables

- **`double R`**
- **`double double sigma`**
- **`double double double T`**
- **`double double double int N`**
- **`double double double int double K`**
- **`double double double int double K1`**
- **`double double double int double double K2`**
- **`double S`**

- double double double double double double **Smin**
- double double double double double double double **Smax**
- double double double double double double double int **imax**
- double double double double double double int int **jmax**
- double double **alfa**
- double double double **beta**

## 7.15.1 Documentation des fonctions

### 7.15.1.1 \_\_declspec()

```
__declspec (
 dllexport)
```

Calcule le prix d'un call européen.

Calcule la grille du delta VL d'un butterfly.

Calcule la grille du prix VL d'un butterfly.

Calcule le delta VL d'un butterfly.

Calcule le prix VL d'un butterfly.

Calcule la grille du delta VL d'un strangle.

Calcule la grille du prix VL d'un strangle.

Calcule le delta VL d'un strangle.

Calcule le prix VL d'un strangle.

Calcule la grille du delta VL d'un bear spread.

Calcule la grille du prix VL d'un bear spread.

Calcule le delta VL d'un bear spread.

Calcule le prix VL d'un bear spread.

Calcule la grille du delta VL d'un bull spread.

Calcule la grille du prix VL d'un bull spread.

Calcule le delta VL d'un bull spread.

Calcule le prix VL d'un bull spread.

Calcule la grille du delta VL d'un double-digital.

Calcule la grille du prix VL d'un double-digital.

Calcule le delta VL d'un double-digital.

Calcule le prix VL d'un double-digital.

Calcule la grille du delta VL d'un put digital.

Calcule la grille du prix VL d'un put digital.

Calcule le delta VL d'un put digital.

Calcule le prix VL d'un put digital.

Calcule la grille du delta VL d'un call digital.

Calcule la grille du prix VL d'un call digital.

Calcule le delta VL d'un call digital.

Calcule le prix VL d'un call digital.

Calcule la grille du delta VL d'un put vanille.

Calcule la grille du prix VL d'un put vanille.

Calcule le delta VL d'un put vanille.

Calcule le prix VL d'un put vanille.

Calcule la grille du delta VL d'un call vanille.

Calcule la grille du prix VL d'un call vanille.

Calcule le delta VL d'un call vanille.

Calcule le prix VL d'un call vanille.

Calcule la volatilité locale.

Calcule la grille du delta BS d'un butterfly.

Calcule la grille du prix BS d'un butterfly.

Calcule le delta BS d'un butterfly.

Calcule le prix BS d'un butterfly.

Calcule la grille du delta BS d'un strangle.

Calcule la grille du prix BS d'un strangle.

Calcule le delta BS d'un strangle.

Calcule le prix BS d'un strangle.

Calcule la grille du delta BS d'un bear spread.

Calcule la grille du prix BS d'un bear spread.

Calcule le delta BS d'un bear spread.

Calcule le prix BS d'un bear spread.

Calcule la grille du delta BS d'un bull spread.

Calcule la grille du prix BS d'un bull spread.

Calcule le delta BS d'un bull spread.

Calcule le prix BS d'un bull spread.

Calcule la grille du delta BS d'un double-digital.

Calcule la grille du prix BS d'un double-digital.

Calcule le delta BS d'un double-digital.

Calcule le prix BS d'un double-digital.

Calcule la grille du delta BS d'un put digital.

Calcule la grille du prix BS d'un put digital.

Calcule le delta BS d'un put digital.

Calcule le prix BS d'un put digital.

Calcule la grille du delta BS d'un call digital.

Calcule la grille du prix BS d'un call digital.

Calcule le delta BS d'un call digital.

Calcule le prix BS d'un call digital.

Calcule la grille du delta BS d'un put vanille.

Calcule la grille du prix BS d'un put vanille.

Calcule le delta BS d'un put vanille.

Calcule le prix BS d'un put vanille.

Calcule la grille du delta BS d'un call vanille.

Calcule la grille du prix BS d'un call vanille.

Calcule le delta BS d'un call vanille.

Calcule le prix BS d'un call vanille.

Calcule le delta Repeated Richardson d'un put américain.

Calcule le prix Repeated Richardson d'un put américain.

Renvoie la matrice des bonds du put américain.

Renvoie la matrice des deltas du put américain.

Renvoie la matrice de valorisation du put américain.

Calcule le delta d'un put américain.

Calcule le prix d'un put américain.

Renvoie la matrice des bonds du call américain.

Renvoie la matrice des deltas du call américain.

Renvoie la matrice de valorisation du call américain.

Calcule le delta d'un call américain.

Calcule le prix d'un call américain.

Calcule le delta MC d'un put sur min.

Calcule le prix MC d'un put sur min.

Renvoie la matrice des bonds du put sur min.

Renvoie la matrice des deltas du put sur min.

Renvoie la matrice de valorisation du put sur min.

Calcule le delta d'un put sur min.

Calcule le prix d'un put sur min.

Calcule le delta MC d'un call sur max.

Calcule le prix MC d'un call sur max.

Renvoie la matrice des bonds du call sur max.

Renvoie la matrice des deltas du call sur max.

Renvoie la matrice de valorisation du call sur max.

Calcule le delta d'un call sur max.

Calcule le prix d'un call sur max.

Calcule le delta MC d'un put sur moyenne géométrique.

Calcule le prix MC d'un put sur moyenne géométrique.

Renvoie la matrice des bonds du put sur moyenne géométrique.

Renvoie la matrice des deltas du put sur moyenne géométrique.

Renvoie la matrice de valorisation du put sur moyenne géométrique.

Calcule le delta d'un put sur moyenne géométrique.

Calcule le prix d'un put sur moyenne géométrique.

Calcule le delta MC d'un call sur moyenne géométrique.

Calcule le prix MC d'un call sur moyenne géométrique.

Renvoie la matrice des bonds du call sur moyenne géométrique.

Renvoie la matrice des deltas du call sur moyenne géométrique.

Renvoie la matrice de valorisation du call sur moyenne géométrique.

Calcule le delta d'un call sur moyenne géométrique.

Calcule le prix d'un call sur moyenne géométrique.

Calcule le delta MC d'un put sur moyenne arithmétique.

Calcule le prix MC d'un put sur moyenne arithmétique.

Renvoie la matrice des bonds du put sur moyenne arithmétique.

Renvoie la matrice des deltas du put sur moyenne arithmétique.

Renvoie la matrice de valorisation du put sur moyenne arithmétique.

Calcule le delta d'un put sur moyenne arithmétique.

Calcule le prix d'un put sur moyenne arithmétique.

Calcule le delta MC d'un call sur moyenne arithmétique.

Calcule le prix MC d'un call sur moyenne arithmétique.

Renvoie la matrice des bonds du call sur moyenne arithmétique.

Renvoie la matrice des deltas du call sur moyenne arithmétique.

Renvoie la matrice de valorisation du call sur moyenne arithmétique.

Calcule le delta d'un call sur moyenne arithmétique.

Calcule le prix d'un call sur moyenne arithmétique.

Renvoie la matrice des bonds du butterfly.

Renvoie la matrice des deltas du butterfly.

Renvoie la matrice de valorisation du butterfly.

Calcule le delta d'une option butterfly.

Calcule le prix d'une option butterfly.

Renvoie la matrice des bonds du strangle.

Renvoie la matrice des deltas du strangle.

Renvoie la matrice de valorisation du strangle.

Calcule le delta d'une option strangle.

Calcule le prix d'une option strangle.

Renvoie la matrice des bonds du bear spread.

Renvoie la matrice des deltas du bear spread.

Renvoie la matrice de valorisation du bear spread.

Calcule le delta d'une option bear spread.

Calcule le prix d'une option bear spread.

Renvoie la matrice des bonds du bull spread.

Renvoie la matrice des deltas du bull spread.

Renvoie la matrice de valorisation du bull spread.

Calcule le delta d'une option bull spread.

Calcule le prix d'une option bull spread.

Renvoie la matrice des bonds du double-digit.

Renvoie la matrice des deltas du double-digit.

Renvoie la matrice de valorisation du double-digit.

Calcule le delta d'une option double-digital.

Calcule le prix d'une option double-digital.

Renvoie la matrice des bonds du put digital.

Renvoie la matrice des deltas du put digital.

Renvoie la matrice de valorisation du put digital.

Calcule le delta d'un put digital.

Calcule le prix d'un put digital.

Renvoie la matrice des bonds du call digital.

Renvoie la matrice des deltas du call digital.

Renvoie la matrice de valorisation du call digital.

Calcule le delta d'un call digital.

Calcule le prix d'un call digital.

Renvoie la matrice des bonds du put vanille.

Renvoie la matrice des deltas du put vanille.

Renvoie la matrice de valorisation du put vanille.

Calcule le delta d'un put européen.

Calcule le prix d'un put européen.

Renvoie la matrice des bonds du call vanille en VARIANT SAFEARRAY.

Renvoie la matrice des deltas du call vanille en VARIANT SAFEARRAY.

Renvoie la matrice 2D V[n][i] du call vanille en VARIANT SAFEARRAY.

Calcule le delta d'un call européen.

## 7.15.2 Documentation des variables

### 7.15.2.1 alfa

```
double double alfa
```

### 7.15.2.2 beta

```
double beta
```

### 7.15.2.3 imax

```
double double double double double double double int imax
```

### 7.15.2.4 jmax

```
double double double double double double double int int jmax
```

### 7.15.2.5 K

```
double double double double K
```

### 7.15.2.6 K1

```
double double double double K1
```

### 7.15.2.7 K2

```
double double double double double K2
```

### 7.15.2.8 N

```
double double double int N
```

### 7.15.2.9 R

```
double double double R
```

### 7.15.2.10 S

```
double S
```

### 7.15.2.11 sigma

```
double double sigma
```

### 7.15.2.12 Smax

```
double double double double double double Smax
```

### 7.15.2.13 Smin

```
double double double double double double Smin
```

### 7.15.2.14 T

```
double double T
```

## 7.16 Exports.h

[Aller à la documentation de ce fichier.](#)

```
00001 #ifndef EXPORTS_H
00002 #define EXPORTS_H
00003
00004 #include "Payoff.h"
00005 #include "Option.h"
00006 #include "European.h"
00007 #include "Aggregator.h"
00008 #include "Asian.h"
00009 #include "American.h"
00010 #include "ParabPDE.h"
00011 #include "Volatility.h"
00012 #include "Diffusion.h"
00013 #include "FDMETHOD.h"
00014 #include "IMPLICITSCHEME.h"
00015 #include "CNMETHOD.h"
00016 #include <windows.h> // MessageBoxA
00017 #include <comdef.h> // VARIANT
00018 #include <OleAuto.h> // SAFEARRAY
00019
00020 extern "C" {
00021
00022 //=====
00023 // Vanilla Call
00024 //=====
00025
00026 __declspec(dllexport) double __stdcall PriceEuCall(
00027 double S0, double R, double sigma, double T, int N, double K
00028);
00029
00030 __declspec(dllexport) double __stdcall DeltaEuCall(
00031 double S0, double R, double sigma, double T, int N, double K
00032);
00033
00034 __declspec(dllexport) VARIANT __stdcall TreeEuCall(
00035 double S0, double R, double sigma, double T, int N, double K
00036);
00037
00038 __declspec(dllexport) VARIANT __stdcall TreeDeltaEuCall(
00039 double S0, double R, double sigma, double T, int N, double K
00040);
00041
00042 __declspec(dllexport) VARIANT __stdcall TreeBondEuCall(
00043 double S0, double R, double sigma, double T, int N, double K
00044);
00045
00046 //=====
00047 //=====
```

```
00062 // Vanilla Put
00063 //=====
00064
00068 __declspec(dllexport) double __stdcall PriceEuPut(
00069 double S0, double R, double sigma, double T, int N, double K
00070);
00071
00075 __declspec(dllexport) double __stdcall DeltaEuPut(
00076 double S0, double R, double sigma, double T, int N, double K
00077);
00078
00082 __declspec(dllexport) VARIANT __stdcall TreeEuPut(
00083 double S0, double R, double sigma, double T, int N, double K
00084);
00085
00089 __declspec(dllexport) VARIANT __stdcall TreeDeltaEuPut(
00090 double S0, double R, double sigma, double T, int N, double K
00091);
00092
00096 __declspec(dllexport) VARIANT __stdcall TreeBondEuPut(
00097 double S0, double R, double sigma, double T, int N, double K
00098);
00099
00100 //=====
00101 // Digital Call
00102 //=====
00103
00107 __declspec(dllexport) double __stdcall PriceDigitCall(
00108 double S0, double R, double sigma, double T, int N, double K
00109);
00110
00114 __declspec(dllexport) double __stdcall DeltaDigitCall(
00115 double S0, double R, double sigma, double T, int N, double K
00116);
00117
00121 __declspec(dllexport) VARIANT __stdcall TreeDigitCall(
00122 double S0, double R, double sigma, double T, int N, double K
00123);
00124
00128 __declspec(dllexport) VARIANT __stdcall TreeDeltaDigitCall(
00129 double S0, double R, double sigma, double T, int N, double K
00130);
00131
00135 __declspec(dllexport) VARIANT __stdcall TreeBondDigitCall(
00136 double S0, double R, double sigma, double T, int N, double K
00137);
00138
00139 //=====
00140 // Digital Put
00141 //=====
00142
00146 __declspec(dllexport) double __stdcall PriceDigitPut(
00147 double S0, double R, double sigma, double T, int N, double K
00148);
00149
00153 __declspec(dllexport) double __stdcall DeltaDigitPut(
00154 double S0, double R, double sigma, double T, int N, double K
00155);
00156
00160 __declspec(dllexport) VARIANT __stdcall TreeDigitPut(
00161 double S0, double R, double sigma, double T, int N, double K
00162);
00163
00167 __declspec(dllexport) VARIANT __stdcall TreeDeltaDigitPut(
00168 double S0, double R, double sigma, double T, int N, double K
00169);
00170
00174 __declspec(dllexport) VARIANT __stdcall TreeBondDigitPut(
00175 double S0, double R, double sigma, double T, int N, double K
00176);
00177
00178 //=====
00179 // Double-Digital
00180 //=====
00181
00185 __declspec(dllexport) double __stdcall PriceDD(
00186 double S0, double R, double sigma, double T, int N, double K1, double K2
00187);
00188
00192 __declspec(dllexport) double __stdcall DeltaDD(
00193 double S0, double R, double sigma, double T, int N, double K1, double K2
00194);
00195
00199 __declspec(dllexport) VARIANT __stdcall TreeDD(
00200 double S0, double R, double sigma, double T, int N, double K1, double K2
00201);
00202
```

```

00206 __declspec(dllexport) VARIANT __stdcall TreeDeltaDD(
00207 double S0, double R, double sigma, double T, int N, double K1, double K2
00208);
00209
00213 __declspec(dllexport) VARIANT __stdcall TreeBondDD(
00214 double S0, double R, double sigma, double T, int N, double K1, double K2
00215);
00216
00217 //=====
00218 // Bull Spread
00219 //=====
00220
00224 __declspec(dllexport) double __stdcall PriceBull(
00225 double S0, double R, double sigma, double T, int N, double K1, double K2
00226);
00227
00231 __declspec(dllexport) double __stdcall DeltaBull(
00232 double S0, double R, double sigma, double T, int N, double K1, double K2
00233);
00234
00238 __declspec(dllexport) VARIANT __stdcall TreeBull(
00239 double S0, double R, double sigma, double T, int N, double K1, double K2
00240);
00241
00245 __declspec(dllexport) VARIANT __stdcall TreeDeltaBull(
00246 double S0, double R, double sigma, double T, int N, double K1, double K2
00247);
00248
00252 __declspec(dllexport) VARIANT __stdcall TreeBondBull(
00253 double S0, double R, double sigma, double T, int N, double K1, double K2
00254);
00255
00256 //=====
00257 // Bear Spread
00258 //=====
00259
00263 __declspec(dllexport) double __stdcall PriceBear(
00264 double S0, double R, double sigma, double T, int N, double K1, double K2
00265);
00266
00270 __declspec(dllexport) double __stdcall DeltaBear(
00271 double S0, double R, double sigma, double T, int N, double K1, double K2
00272);
00273
00277 __declspec(dllexport) VARIANT __stdcall TreeBear(
00278 double S0, double R, double sigma, double T, int N, double K1, double K2
00279);
00280
00284 __declspec(dllexport) VARIANT __stdcall TreeDeltaBear(
00285 double S0, double R, double sigma, double T, int N, double K1, double K2
00286);
00287
00291 __declspec(dllexport) VARIANT __stdcall TreeBondBear(
00292 double S0, double R, double sigma, double T, int N, double K1, double K2
00293);
00294
00295 //=====
00296 // Strangle
00297 //=====
00298
00302 __declspec(dllexport) double __stdcall PriceStrangle(
00303 double S0, double R, double sigma, double T, int N, double K1, double K2
00304);
00305
00309 __declspec(dllexport) double __stdcall DeltaStrangle(
00310 double S0, double R, double sigma, double T, int N, double K1, double K2
00311);
00312
00316 __declspec(dllexport) VARIANT __stdcall TreeStrangle(
00317 double S0, double R, double sigma, double T, int N, double K1, double K2
00318);
00319
00323 __declspec(dllexport) VARIANT __stdcall TreeDeltaStrangle(
00324 double S0, double R, double sigma, double T, int N, double K1, double K2
00325);
00326
00330 __declspec(dllexport) VARIANT __stdcall TreeBondStrangle(
00331 double S0, double R, double sigma, double T, int N, double K1, double K2
00332);
00333
00334 //=====
00335 // Butterfly
00336 //=====
00337
00341 __declspec(dllexport) double __stdcall PriceButterfly(
00342 double S0, double R, double sigma, double T, int N, double K1, double K2
00343);

```

```
00344
00348 __declspec(dllexport) double __stdcall DeltaButterfly(
00349 double S0, double R, double sigma, double T, int N, double K1, double K2
00350);
00351
00355 __declspec(dllexport) VARIANT __stdcall TreeButterfly(
00356 double S0, double R, double sigma, double T, int N, double K1, double K2
00357);
00358
00362 __declspec(dllexport) VARIANT __stdcall TreeDeltaButterfly(
00363 double S0, double R, double sigma, double T, int N, double K1, double K2
00364);
00365
00369 __declspec(dllexport) VARIANT __stdcall TreeBondButterfly(
00370 double S0, double R, double sigma, double T, int N, double K1, double K2
00371);
00372
00373 //=====
00374 // Arithmetic Call
00375 //=====
00376
00380 __declspec(dllexport) double __stdcall PriceAritCall(
00381 double S0, double R, double sigma, double T, int N, double K
00382);
00383
00387 __declspec(dllexport) double __stdcall DeltaAritCall(
00388 double S0, double R, double sigma, double T, int N, double K
00389);
00390
00394 __declspec(dllexport) VARIANT __stdcall TreeAritCall(
00395 double S0, double R, double sigma, double T, int N, double K
00396);
00397
00401 __declspec(dllexport) VARIANT __stdcall TreeDeltaAritCall(
00402 double S0, double R, double sigma, double T, int N, double K
00403);
00404
00408 __declspec(dllexport) VARIANT __stdcall TreeBondAritCall(
00409 double S0, double R, double sigma, double T, int N, double K
00410);
00411
00415 __declspec(dllexport) double __stdcall PriceAritCallMC(
00416 double S0, double R, double sigma, double T, int N, double K
00417);
00418
00422 __declspec(dllexport) double __stdcall DeltaAritCallMC(
00423 double S0, double R, double sigma, double T, int N, double K
00424);
00425
00426 //=====
00427 // Arithmetic Put
00428 //=====
00429
00433 __declspec(dllexport) double __stdcall PriceAritPut(
00434 double S0, double R, double sigma, double T, int N, double K
00435);
00436
00440 __declspec(dllexport) double __stdcall DeltaAritPut(
00441 double S0, double R, double sigma, double T, int N, double K
00442);
00443
00447 __declspec(dllexport) VARIANT __stdcall TreeAritPut(
00448 double S0, double R, double sigma, double T, int N, double K
00449);
00450
00454 __declspec(dllexport) VARIANT __stdcall TreeDeltaAritPut(
00455 double S0, double R, double sigma, double T, int N, double K
00456);
00457
00461 __declspec(dllexport) VARIANT __stdcall TreeBondAritPut(
00462 double S0, double R, double sigma, double T, int N, double K
00463);
00464
00468 __declspec(dllexport) double __stdcall PriceAritPutMC(
00469 double S0, double R, double sigma, double T, int N, double K
00470);
00471
00475 __declspec(dllexport) double __stdcall DeltaAritPutMC(
00476 double S0, double R, double sigma, double T, int N, double K
00477);
00478
00479 //=====
00480 // Geometric Call
00481 //=====
00482
00486 __declspec(dllexport) double __stdcall PriceGeomCall(
00487 double S0, double R, double sigma, double T, int N, double K
```

```

00488);
00489
00493 __declspec(dllexport) double __stdcall DeltaGeomCall(
00494 double S0, double R, double sigma, double T, int N, double K
00495);
00496
00500 __declspec(dllexport) VARIANT __stdcall TreeGeomCall(
00501 double S0, double R, double sigma, double T, int N, double K
00502);
00503
00507 __declspec(dllexport) VARIANT __stdcall TreeDeltaGeomCall(
00508 double S0, double R, double sigma, double T, int N, double K
00509);
00510
00514 __declspec(dllexport) VARIANT __stdcall TreeBondGeomCall(
00515 double S0, double R, double sigma, double T, int N, double K
00516);
00517
00521 __declspec(dllexport) double __stdcall PriceGeomCallMC(
00522 double S0, double R, double sigma, double T, int N, double K
00523);
00524
00528 __declspec(dllexport) double __stdcall DeltaGeomCallMC(
00529 double S0, double R, double sigma, double T, int N, double K
00530);
00531
00532 //=====
00533 // Geometric Put
00534 //=====
00535
00539 __declspec(dllexport) double __stdcall PriceGeomPut(
00540 double S0, double R, double sigma, double T, int N, double K
00541);
00542
00546 __declspec(dllexport) double __stdcall DeltaGeomPut(
00547 double S0, double R, double sigma, double T, int N, double K
00548);
00549
00553 __declspec(dllexport) VARIANT __stdcall TreeGeomPut(
00554 double S0, double R, double sigma, double T, int N, double K
00555);
00556
00560 __declspec(dllexport) VARIANT __stdcall TreeDeltaGeomPut(
00561 double S0, double R, double sigma, double T, int N, double K
00562);
00563
00567 __declspec(dllexport) VARIANT __stdcall TreeBondGeomPut(
00568 double S0, double R, double sigma, double T, int N, double K
00569);
00570
00574 __declspec(dllexport) double __stdcall PriceGeomPutMC(
00575 double S0, double R, double sigma, double T, int N, double K
00576);
00577
00581 __declspec(dllexport) double __stdcall DeltaGeomPutMC(
00582 double S0, double R, double sigma, double T, int N, double K
00583);
00584
00585 //=====
00586 // Lookback Call
00587 //=====
00588
00592 __declspec(dllexport) double __stdcall PriceMaxCall(
00593 double S0, double R, double sigma, double T, int N, double K
00594);
00595
00599 __declspec(dllexport) double __stdcall DeltaMaxCall(
00600 double S0, double R, double sigma, double T, int N, double K
00601);
00602
00606 __declspec(dllexport) VARIANT __stdcall TreeMaxCall(
00607 double S0, double R, double sigma, double T, int N, double K
00608);
00609
00613 __declspec(dllexport) VARIANT __stdcall TreeDeltaMaxCall(
00614 double S0, double R, double sigma, double T, int N, double K
00615);
00616
00620 __declspec(dllexport) VARIANT __stdcall TreeBondMaxCall(
00621 double S0, double R, double sigma, double T, int N, double K
00622);
00623
00627 __declspec(dllexport) double __stdcall PriceMaxCallMC(
00628 double S0, double R, double sigma, double T, int N, double K
00629);
00630
00634 __declspec(dllexport) double __stdcall DeltaMaxCallMC(

```

```
00635 double S0, double R, double sigma, double T, int N, double K
00636);
00637
00638 //=====
00639 // Lookback Put
00640 //=====
00641
00642 __declspec(dllexport) double __stdcall PriceMinPut(
00643 double S0, double R, double sigma, double T, int N, double K
00644);
00645
00646 __declspec(dllexport) double __stdcall DeltaMinPut(
00647 double S0, double R, double sigma, double T, int N, double K
00648);
00649
00650 __declspec(dllexport) VARIANT __stdcall TreeMinPut(
00651 double S0, double R, double sigma, double T, int N, double K
00652);
00653
00654
00655 __declspec(dllexport) VARIANT __stdcall TreeDeltaMinPut(
00656 double S0, double R, double sigma, double T, int N, double K
00657);
00658
00659 __declspec(dllexport) VARIANT __stdcall TreeBondMinPut(
00660 double S0, double R, double sigma, double T, int N, double K
00661);
00662
00663 __declspec(dllexport) VARIANT __stdcall PriceMinPutMC(
00664 double S0, double R, double sigma, double T, int N, double K
00665);
00666
00667 __declspec(dllexport) VARIANT __stdcall DeltaMinPutMC(
00668 double S0, double R, double sigma, double T, int N, double K
00669);
00670
00671 __declspec(dllexport) double __stdcall PriceAmCall(
00672 double S0, double R, double sigma, double T, int N, double K
00673);
00674
00675 __declspec(dllexport) double __stdcall DeltaAmCall(
00676 double S0, double R, double sigma, double T, int N, double K
00677);
00678
00679 __declspec(dllexport) VARIANT __stdcall TreeAmCall(
00680 double S0, double R, double sigma, double T, int N, double K
00681);
00682
00683 __declspec(dllexport) VARIANT __stdcall TreeDeltaAmCall(
00684 double S0, double R, double sigma, double T, int N, double K
00685);
00686
00687 __declspec(dllexport) VARIANT __stdcall TreeBondAmCall(
00688 double S0, double R, double sigma, double T, int N, double K
00689);
00690
00691 //=====
00692 // American Call
00693 //=====
00694
00695 __declspec(dllexport) double __stdcall PriceAmPut(
00696 double S0, double R, double sigma, double T, int N, double K
00697);
00698
00699 __declspec(dllexport) double __stdcall DeltaAmPut(
00700 double S0, double R, double sigma, double T, int N, double K
00701);
00702
00703 __declspec(dllexport) VARIANT __stdcall TreeAmPut(
00704 double S0, double R, double sigma, double T, int N, double K
00705);
00706
00707 __declspec(dllexport) VARIANT __stdcall TreeDeltaAmPut(
00708 double S0, double R, double sigma, double T, int N, double K
00709);
00710
00711 __declspec(dllexport) VARIANT __stdcall TreeBondAmPut(
00712 double S0, double R, double sigma, double T, int N, double K
00713);
00714
00715
00716 __declspec(dllexport) double __stdcall PriceAmPutRR(
00717 double S0, double R, double sigma, double T, int N, double K
00718);
00719
00720 __declspec(dllexport) double __stdcall DeltaAmPutRR(
00721 double S0, double R, double sigma, double T, int N, double K
00722);
00723
00724 __declspec(dllexport) VARIANT __stdcall TreeAmPutRR(
00725 double S0, double R, double sigma, double T, int N, double K
00726);
00727
00728 __declspec(dllexport) VARIANT __stdcall TreeDeltaAmPutRR(
00729 double S0, double R, double sigma, double T, int N, double K
00730);
00731
00732 //=====
00733 // American Put
00734 //=====
00735
00736 __declspec(dllexport) double __stdcall PriceAmPut(
00737 double S0, double R, double sigma, double T, int N, double K
00738);
00739
00740 __declspec(dllexport) double __stdcall DeltaAmPut(
00741 double S0, double R, double sigma, double T, int N, double K
00742);
00743
00744 __declspec(dllexport) VARIANT __stdcall TreeAmPut(
00745 double S0, double R, double sigma, double T, int N, double K
00746);
00747
00748 __declspec(dllexport) VARIANT __stdcall TreeDeltaAmPut(
00749 double S0, double R, double sigma, double T, int N, double K
00750);
00751
00752 __declspec(dllexport) VARIANT __stdcall TreeBondAmPut(
00753 double S0, double R, double sigma, double T, int N, double K
00754);
00755
00756 __declspec(dllexport) VARIANT __stdcall TreeAmPutRR(
00757 double S0, double R, double sigma, double T, int N, double K
00758);
00759
00760 __declspec(dllexport) VARIANT __stdcall TreeDeltaAmPutRR(
00761 double S0, double R, double sigma, double T, int N, double K
00762);
00763
00764 __declspec(dllexport) VARIANT __stdcall TreeBondAmPutRR(
00765 double S0, double R, double sigma, double T, int N, double K
00766);
00767
00768
00769 __declspec(dllexport) double __stdcall PriceAmPutRR(
00770 double S0, double R, double sigma, double T, int N, double K
00771);
00772
00773 __declspec(dllexport) double __stdcall DeltaAmPutRR(
00774 double S0, double R, double sigma, double T, int N, double K
00775);
```

```

00779 __declspec(dllexport) double __stdcall DeltaAmPutRR(
00780 double S0, double R, double sigma, double T, int N, double K
00781);
00782
00783 //=====
00784 // Black-Scholes
00785 //=====
00786
00787 __declspec(dllexport) double __stdcall PriceEuCallBS(
00788 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00789 imax, int jmax
00790);
00791
00792 __declspec(dllexport) double __stdcall DeltaEuCallBS(
00793 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00794 imax, int jmax
00795);
00796
00797 __declspec(dllexport) VARIANT __stdcall GridPriceEuCallBS(
00798 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00799);
00800
00801 __declspec(dllexport) VARIANT __stdcall GridDeltaEuCallBS(
00802 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00803);
00804
00805 __declspec(dllexport) double __stdcall PriceEuPutBS(
00806 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00807 imax, int jmax
00808);
00809
00810 __declspec(dllexport) double __stdcall DeltaEuPutBS(
00811 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00812 imax, int jmax
00813);
00814
00815 __declspec(dllexport) VARIANT __stdcall GridPriceEuPutBS(
00816 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00817);
00818
00819 __declspec(dllexport) VARIANT __stdcall GridDeltaEuPutBS(
00820 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00821);
00822
00823 __declspec(dllexport) double __stdcall PriceDigitCallBS(
00824 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00825 imax, int jmax
00826);
00827
00828 __declspec(dllexport) double __stdcall DeltaDigitCallBS(
00829 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00830 imax, int jmax
00831);
00832
00833 __declspec(dllexport) VARIANT __stdcall GridPriceDigitCallBS(
00834 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00835);
00836
00837 __declspec(dllexport) VARIANT __stdcall GridDeltaDigitCallBS(
00838 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00839);
00840
00841 __declspec(dllexport) double __stdcall PriceDigitPutBS(
00842 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00843 imax, int jmax
00844);
00845
00846 __declspec(dllexport) double __stdcall DeltaDigitPutBS(
00847 double t, double S, double sigma, double T, double R, double K, double Smin, double Smax, int
00848 imax, int jmax
00849);
00850
00851 __declspec(dllexport) VARIANT __stdcall GridPriceDigitPutBS(
00852 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00853);
00854
00855 __declspec(dllexport) VARIANT __stdcall GridDeltaDigitPutBS(
00856 double sigma, double T, double R, double K, double Smin, double Smax, int imax, int jmax
00857);
00858
00859 __declspec(dllexport) double __stdcall PriceDDBS(
00860 double t, double S, double sigma, double T, double R, double K1,
00861 double K2, double Smin, double Smax, int imax, int jmax
00862);
00863
00864 __declspec(dllexport) double __stdcall DeltaDDBS(
00865 double t, double S, double sigma, double T, double R, double K1,
00866);

```

```
00912 double K2, double Smin, double Smax, int imax, int jmax
00913);
00914
00918 __declspec(dllexport) VARIANT __stdcall GridPriceDDBS(
00919 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
00920 int jmax
00921);
00922
00925 __declspec(dllexport) VARIANT __stdcall GridDeltaDDBS(
00926 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
00927 int jmax
00928);
00929
00932 __declspec(dllexport) double __stdcall PriceBullBS(
00933 double t, double S, double sigma, double T, double R, double K1,
00934 double K2, double Smin, double Smax, int imax, int jmax
00935);
00936
00940 __declspec(dllexport) double __stdcall DeltaBullBS(
00941 double t, double S, double sigma, double T, double R, double K1,
00942 double K2, double Smin, double Smax, int imax, int jmax
00943);
00944
00948 __declspec(dllexport) VARIANT __stdcall GridPriceBullBS(
00949 double sigma, double T, double K1, double K2, double Smin, double Smax, int imax,
00950 int jmax
00951);
00955
00956 __declspec(dllexport) VARIANT __stdcall GridDeltaBullBS(
00957 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
00958 int jmax
00959);
00962 __declspec(dllexport) double __stdcall PriceBearBS(
00963 double t, double S, double sigma, double T, double R, double K1,
00964 double K2, double Smin, double Smax, int imax, int jmax
00965);
00966
00970 __declspec(dllexport) double __stdcall DeltaBearBS(
00971 double t, double S, double sigma, double T, double R, double K1,
00972 double K2, double Smin, double Smax, int imax, int jmax
00973);
00974
00978 __declspec(dllexport) VARIANT __stdcall GridPriceBearBS(
00979 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
00980 int jmax
00981);
00985
00986 __declspec(dllexport) VARIANT __stdcall GridDeltaBearBS(
00987 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
00988 int jmax
00989);
00992 __declspec(dllexport) double __stdcall PriceStrangleBS(
00993 double t, double S, double sigma, double T, double R, double K1,
00994 double K2, double Smin, double Smax, int imax, int jmax
00995);
00996
01000 __declspec(dllexport) double __stdcall DeltaStrangleBS(
01001 double t, double S, double sigma, double T, double R, double K1,
01002 double K2, double Smin, double Smax, int imax, int jmax
01003);
01004
01008 __declspec(dllexport) VARIANT __stdcall GridPriceStrangleBS(
01009 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
01010 int jmax
01011);
01015
01016 __declspec(dllexport) VARIANT __stdcall GridDeltaStrangleBS(
01017 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
01018 int jmax
01019);
01022 __declspec(dllexport) double __stdcall PriceButterflyBS(
01023 double t, double S, double sigma, double T, double R, double K1,
01024 double K2, double Smin, double Smax, int imax, int jmax
01025);
01026
01030 __declspec(dllexport) double __stdcall DeltaButterflyBS(
01031 double t, double S, double sigma, double T, double R, double K1,
01032 double K2, double Smin, double Smax, int imax, int jmax
01033);
01034
01038 __declspec(dllexport) VARIANT __stdcall GridPriceButterflyBS(
01039 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
01040 int jmax
01041);
```

```

01041
01045 __declspec(dllexport) VARIANT __stdcall GridDeltaButterflyBS(
01046 double sigma, double T, double R, double K1, double K2, double Smin, double Smax, int imax,
01047 int jmax
01048);
01049 //=====
01050 // Volatilité Locale
01051 //=====
01052
01056 __declspec(dllexport) double __stdcall Vol(
01057 double t, double S, double alfa, double beta
01058);
01059
01063 __declspec(dllexport) double __stdcall PriceEuCallVL(
01064 double t, double S, double alfa, double beta, double T, double R,
01065 double K, double Smin, double Smax, int imax, int jmax
01066);
01067
01071 __declspec(dllexport) double __stdcall DeltaEuCallVL(
01072 double t, double S, double alfa, double beta, double T, double R,
01073 double K, double Smin, double Smax, int imax, int jmax
01074);
01075
01079 __declspec(dllexport) VARIANT __stdcall GridPriceEuCallVL(
01080 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01081 int jmax
01082);
01086 __declspec(dllexport) VARIANT __stdcall GridDeltaEuCallVL(
01087 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01088 int jmax
01089);
01093 __declspec(dllexport) double __stdcall PriceEuPutVL(
01094 double t, double S, double alfa, double beta, double T, double R,
01095 double K, double Smin, double Smax, int imax, int jmax
01096);
01097
01101 __declspec(dllexport) double __stdcall DeltaEuPutVL(
01102 double t, double S, double alfa, double beta, double T, double R,
01103 double K, double Smin, double Smax, int imax, int jmax
01104);
01105
01109 __declspec(dllexport) VARIANT __stdcall GridPriceEuPutVL(
01110 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01111 int jmax
01112);
01116 __declspec(dllexport) VARIANT __stdcall GridDeltaEuPutVL(
01117 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01118 int jmax
01119);
01123 __declspec(dllexport) double __stdcall PriceDigitCallVL(
01124 double t, double S, double alfa, double beta, double T, double R,
01125 double K, double Smin, double Smax, int imax, int jmax
01126);
01127
01131 __declspec(dllexport) double __stdcall DeltaDigitCallVL(
01132 double t, double S, double alfa, double beta, double T, double R,
01133 double K, double Smin, double Smax, int imax, int jmax
01134);
01135
01139 __declspec(dllexport) VARIANT __stdcall GridPriceDigitCallVL(
01140 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01141 int jmax
01142);
01146 __declspec(dllexport) VARIANT __stdcall GridDeltaDigitCallVL(
01147 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01148 int jmax
01149);
01153 __declspec(dllexport) double __stdcall PriceDigitPutVL(
01154 double t, double S, double alfa, double beta, double T, double R,
01155 double K, double Smin, double Smax, int imax, int jmax
01156);
01157
01161 __declspec(dllexport) double __stdcall DeltaDigitPutVL(
01162 double t, double S, double alfa, double beta, double T, double R,
01163 double K, double Smin, double Smax, int imax, int jmax
01164);
01165
01169 __declspec(dllexport) VARIANT __stdcall GridPriceDigitPutVL(
01170 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01171 int jmax

```

```
01171);
01172
01176 __declspec(dllexport) VARIANT __stdcall GridDeltaDigitPutVL(
01177 double alfa, double beta, double T, double R, double K, double Smin, double Smax, int imax,
01178 int jmax
01179);
01179
01183 __declspec(dllexport) double __stdcall PriceDDVL(
01184 double t, double S, double alfa, double beta, double T, double R,
01185 double K1, double K2, double Smin, double Smax, int imax, int jmax
01186);
01187
01191 __declspec(dllexport) double __stdcall DeltaDDVL(
01192 double t, double S, double alfa, double beta, double T, double R,
01193 double K1, double K2, double Smin, double Smax, int imax, int jmax
01194);
01195
01199 __declspec(dllexport) VARIANT __stdcall GridPriceDDVL(
01200 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01201 int imax, int jmax
01202);
01206 __declspec(dllexport) VARIANT __stdcall GridDeltaDDVL(
01207 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01208 int imax, int jmax
01209);
01213 __declspec(dllexport) double __stdcall PriceBullVL(
01214 double t, double S, double alfa, double beta, double T, double R,
01215 double K1, double K2, double Smin, double Smax, int imax, int jmax
01216);
01217
01221 __declspec(dllexport) double __stdcall DeltaBullVL(
01222 double t, double S, double alfa, double beta, double T, double R,
01223 double K1, double K2, double Smin, double Smax, int imax, int jmax
01224);
01225
01229 __declspec(dllexport) VARIANT __stdcall GridPriceBullVL(
01230 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01231 int imax, int jmax
01232);
01236 __declspec(dllexport) VARIANT __stdcall GridDeltaBullVL(
01237 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01238 int imax, int jmax
01239);
01243 __declspec(dllexport) double __stdcall PriceBearVL(
01244 double t, double S, double alfa, double beta, double T, double R,
01245 double K1, double K2, double Smin, double Smax, int imax, int jmax
01246);
01247
01251 __declspec(dllexport) double __stdcall DeltaBearVL(
01252 double t, double S, double alfa, double beta, double T, double R,
01253 double K1, double K2, double Smin, double Smax, int imax, int jmax
01254);
01255
01259 __declspec(dllexport) VARIANT __stdcall GridPriceBearVL(
01260 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01261 int imax, int jmax
01262);
01266 __declspec(dllexport) VARIANT __stdcall GridDeltaBearVL(
01267 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01268 int imax, int jmax
01269);
01273 __declspec(dllexport) double __stdcall PriceStrangleVL(
01274 double t, double S, double alfa, double beta, double T, double R,
01275 double K1, double K2, double Smin, double Smax, int imax, int jmax
01276);
01277
01281 __declspec(dllexport) double __stdcall DeltaStrangleVL(
01282 double t, double S, double alfa, double beta, double T, double R,
01283 double K1, double K2, double Smin, double Smax, int imax, int jmax
01284);
01285
01289 __declspec(dllexport) VARIANT __stdcall GridPriceStrangleVL(
01290 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01291 int imax, int jmax
01292);
01296 __declspec(dllexport) VARIANT __stdcall GridDeltaStrangleVL(
01297 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01298 int imax, int jmax
01299);
```

```

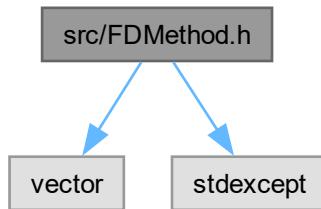
01303 __declspec(dllexport) double __stdcall PriceButterflyVL(
01304 double t, double S, double alfa, double beta, double T, double R,
01305 double K1, double K2, double Smin, double Smax, int imax, int jmax
01306);
01307
01311 __declspec(dllexport) double __stdcall DeltaButterflyVL(
01312 double t, double S, double alfa, double beta, double T, double R,
01313 double K1, double K2, double Smin, double Smax, int imax, int jmax
01314);
01315
01319 __declspec(dllexport) VARIANT __stdcall GridPriceButterflyVL(
01320 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01321 int imax, int jmax
01322);
01326 __declspec(dllexport) VARIANT __stdcall GridDeltaButterflyVL(
01327 double alfa, double beta, double T, double R, double K1, double K2, double Smin, double Smax,
01328 int imax, int jmax
01329);
01330 } // extern "C"
01331
01332 #endif // EXPORTS_H
01333
01334
01335

```

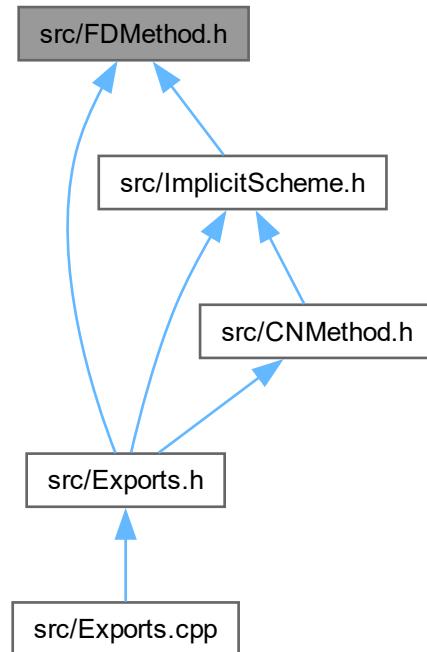
## 7.17 Référence du fichier src/FDMethod.h

```
#include <vector>
#include <stdexcept>
```

Graphe des dépendances par inclusion de FDMethod.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Classes

- class [pde::FDMethod< TPDE >](#)

*Méthodes aux différences finies pour la résolution numérique d'EDP.*

- struct [pde::FDMethod< TPDE >::Grid](#)

*Grille : prix et delta.*

## Espaces de nommage

- namespace [pde](#)

## 7.18 FDMethod.h

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef FDMETHOD_H
00002 #define FDMETHOD_H
00003
00004 #include <vector>
00005 #include <stdexcept>
00006

```

```

00007 namespace pde{
00008
00013 template<typename TPDE>
00014 class FDMETHOD {
00015 protected:
00016 TPDE pde_;
00017 int imax_, jmax_;
00018 double dt_, ds_;
00019 std::vector<std::vector<double>> V;
00020
00021 public:
00022 FDMETHOD(const TPDE& pde, int imax, int jmax);
00023
00024 double t(double i) const { return dt_ * i; }
00025 double S(int j) const { return pde_.Smin() + ds_ * j; }
00026
00027 double a(double i, int j) const { return pde_.a(t(i), S(j)); }
00028 double b(double i, int j) const { return pde_.b(t(i), S(j)); }
00029 double c(double i, int j) const { return pde_.c(t(i), S(j)); }
00030 double d(double i, int j) const { return pde_.d(t(i), S(j)); }
00031
00032 double f(int j) const { return pde_.Terminal(S(j)); }
00033 double fu(int i) const { return pde_.Upper(t(i)); }
00034 double fl(int i) const { return pde_.Lower(t(i)); }
00035
00036 double v(double t, double S) const;
00037
00038 double delta(double t, double S) const;
00039
00040 struct Grid {
00041 std::vector<std::vector<double>> val;
00042 std::vector<std::vector<double>> del;
00043 };
00044
00045 Grid grid() const;
00046 };
00047
00048 template<typename TPDE>
00049 FDMETHOD<TPDE>::FDMETHOD(const TPDE& pde, int imax, int jmax)
00050 : pde_(pde), imax_(imax), jmax_(jmax)
00051 {
00052 if (imax_ <= 0) throw std::invalid_argument("Nt doit être >= 1");
00053 if (jmax_ <= 1) throw std::invalid_argument("NS doit être >= 2");
00054
00055 ds_ = (pde_.Smax() - pde_.Smin()) / jmax_;
00056 dt_ = pde_.T() / imax_;
00057 V.resize(imax_ + 1);
00058 for (int i = 0; i <= imax_; i++)
00059 V[i].resize(jmax_ + 1);
00060 }
00061
00062 template<typename TPDE>
00063 double FDMETHOD<TPDE>::v(double t, double S) const {
00064 if (t < 0 || t > pde_.T())
00065 throw std::out_of_range("t hors du domaine");
00066 if (S < pde_.Smin() || S > pde_.Smax())
00067 throw std::invalid_argument("S hors du domaine");
00068 if (S == pde_.Smax())
00069 return pde_.Upper(t);
00070 if (t == pde_.T())
00071 return pde_.Terminal(S);
00072
00073 int i = (int)(t / dt_);
00074 int j = (int)((S - pde_.Smin()) / ds_);
00075 double l1 = (t - FDMETHOD<TPDE>::t(i)) / dt_, l0 = 1.0 - l1;
00076 double wl = (S - FDMETHOD<TPDE>::S(j)) / ds_, w0 = 1.0 - wl;
00077 return l1 * wl * V[i + 1][j + 1] + l1 * w0 * V[i + 1][j]
00078 + l0 * wl * V[i][j + 1] + l0 * w0 * V[i][j];
00079 }
00080
00081 template<typename TPDE>
00082 double FDMETHOD<TPDE>::delta(double t, double S) const {
00083 if (t < 0 || t > pde_.T())
00084 throw std::out_of_range("t hors du domaine");
00085 if (S < pde_.Smin() || S > pde_.Smax())
00086 throw std::invalid_argument("S hors du domaine");
00087
00088 double dlt;
00089 int i = (int)(t / dt_);
00090 int j = (int)((S - pde_.Smin()) / ds_);
00091 if (j == 0)
00092 dlt = (V[i][1] - V[i][0]) / ds_;
00093 else if (j == jmax_)
00094 dlt = (V[i][jmax_] - V[i][jmax_ - 1]) / ds_;
00095 else
00096 dlt = (V[i][j + 1] - V[i][j - 1]) / (2.0 * ds_);
00097 return std::min<double>(1.0, std::max<double>(-1.0, dlt));
00098 }
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112

```

```

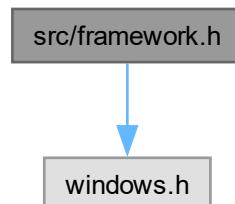
00113 }
00114
00115 template<typename TPDE>
00116 typename FDMethod<TPDE>::Grid FDMethod<TPDE>::grid() const {
00117 int N = 10;
00118 std::vector<double> tvals(N + 1), Svals(N + 1);
00119 for (int i = 0; i <= N; ++i)
00120 tvals[i] = pde_.T() * i / double(N);
00121 for (int j = 0; j <= N; ++j)
00122 Svals[j] = pde_.Smin() + (pde_.Smax() - pde_.Smin()) * j / double(N);
00123
00124 Grid M;
00125 M.val.assign(N + 1, std::vector<double>(N + 1));
00126 M.del.assign(N + 1, std::vector<double>(N + 1));
00127 for (int i = 0; i <= N; ++i) {
00128 for (int j = 0; j <= N; ++j) {
00129 M.val[i][j] = v(tvals[i], Svals[j]);
00130 M.del[i][j] = delta(tvals[i], Svals[j]);
00131 }
00132 }
00133 return M;
00134 }
00135
00136 } // namespace pde
00137
00138 #endif // FDMETHOD_H
00139
00140

```

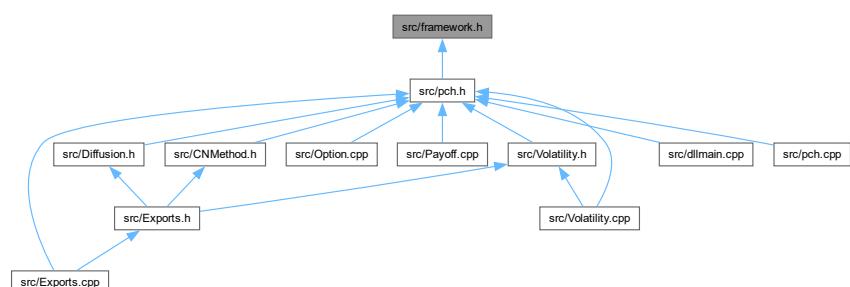
## 7.19 Référence du fichier src/framework.h

#include <windows.h>

Graphe des dépendances par inclusion de framework.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Macros

— `#define WIN32_LEAN_AND_MEAN`

### 7.19.1 Documentation des macros

#### 7.19.1.1 WIN32\_LEAN\_AND\_MEAN

```
#define WIN32_LEAN_AND_MEAN
```

## 7.20 framework.h

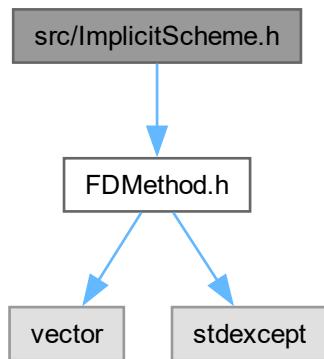
[Aller à la documentation de ce fichier.](#)

```
00001 #pragma once
00002
00003 #define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
00004 // Windows Header Files
00005 #include <windows.h>
```

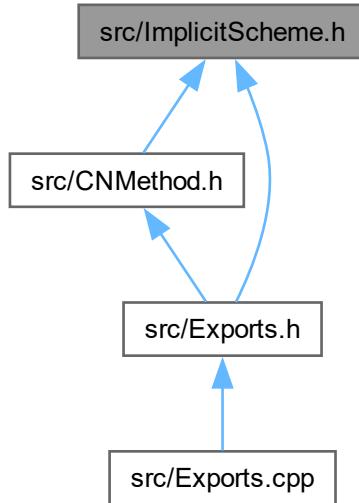
## 7.21 Référence du fichier src/ImplicitScheme.h

```
#include "FDMethod.h"
```

Graphe des dépendances par inclusion de ImplicitScheme.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Classes

- class `pde::ImplicitScheme< TPDE >`

*Schéma implicite aux différences finies.*

## Espaces de nommage

- namespace `pde`

## 7.22 ImplicitScheme.h

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef IMPLICITSCHHEME_H
00002 #define IMPLICITSCHHEME_H
00003
00004 #include "FDMethod.h"
00005
00006 namespace pde {
00007
00012 template<typename TPDE>
00013 class ImplicitScheme : public FDMethod<TPDE> {
00014 public:
00015 ImplicitScheme(const TPDE& pde, int imax, int jmax);
00016
00020 virtual double A(int i, int j) const = 0;
00021 virtual double B(int i, int j) const = 0;
00022 virtual double C(int i, int j) const = 0;
00023 virtual double D(int i, int j) const = 0;
00024 virtual double E(int i, int j) const = 0;
00025 virtual double F(int i, int j) const = 0;

```

```

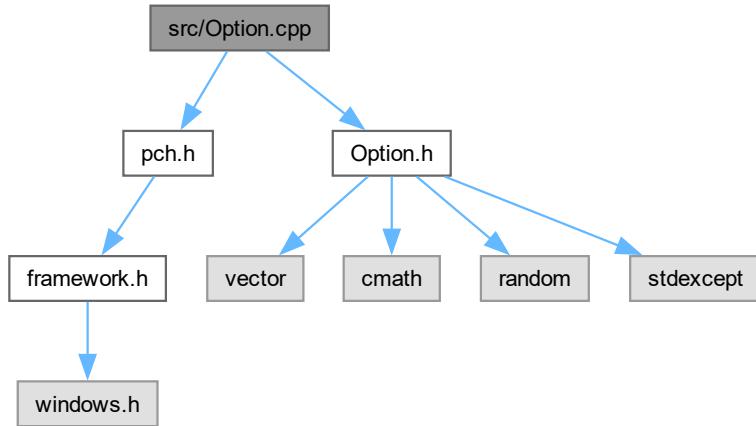
00026 virtual double G(int i, int j) const = 0;
00027
00032 std::vector<double> LUdecomposition(int i, std::vector<double> q) const;
00033
00037 void SolvePDE();
00038
00039 std::vector<double> w(int i) const;
00040 std::vector<double> A(int i, std::vector<double> q) const;
00041 };
00042
00043 template<typename TPDE>
00044 ImplicitScheme<TPDE>::ImplicitScheme(const TPDE& pde, int imax, int jmax)
00045 : FDMMethod<TPDE>(pde, imax, jmax)
00046 {
00047 }
00048
00049 template<typename TPDE>
00050 std::vector<double> ImplicitScheme<TPDE>::w(int i) const {
00051 std::vector<double> w(this->jmax_ + 1);
00052 w[1] = D(i, 1) + A(i, 1) * this->f1(i - 1) - E(i, 1) * this->f1(i - 1);
00053 for (int j = 2; j < this->jmax_ - 1; j++)
00054 w[j] = D(i, j);
00055 w[this->jmax_ - 1] = D(i, this->jmax_ - 1) + C(i, this->jmax_ - 1)
00056 * this->fu(i) - G(i, this->jmax_ - 1) * this->fu(i - 1);
00057 return w;
00058 }
00059
00060 template<typename TPDE>
00061 std::vector<double> ImplicitScheme<TPDE>::A(int i, std::vector<double> q) const {
00062 std::vector<double> p(this->jmax_ + 1);
00063 p[1] = B(i, 1) * q[1] + C(i, 1) * q[2];
00064 for (int j = 2; j < this->jmax_ - 1; j++)
00065 p[j] = A(i, j) * q[j - 1] + B(i, j) * q[j] + C(i, j) * q[j + 1];
00066 p[this->jmax_ - 1] = A(i, this->jmax_ - 1) * q[this->jmax_ - 2] + B(i, this->jmax_ - 1) *
00067 q[this->jmax_ - 1];
00068 return p;
00069 }
00070
00071 template<typename TPDE>
00072 std::vector<double> ImplicitScheme<TPDE>::LUdecomposition(int i, std::vector<double> q) const {
00073 std::vector<double> p(this->jmax_ + 1), r(this->jmax_ + 1), y(this->jmax_ + 1);
00074 r[1] = F(i, 1);
00075 y[1] = q[1];
00076 for (int j = 2; j < this->jmax_; j++) {
00077 r[j] = F(i, j) - E(i, j) * G(i, j - 1) / r[j - 1];
00078 y[j] = q[j] - E(i, j) * y[j - 1] / r[j - 1];
00079 }
00080 p[this->jmax_ - 1] = y[this->jmax_ - 1] / r[this->jmax_ - 1];
00081
00082 for (int j = this->jmax_ - 2; j > 0; j--)
00083 p[j] = (y[j] - G(i, j) * p[j + 1]) / r[j];
00084 return p;
00085 }
00086
00087 template<typename TPDE>
00088 void ImplicitScheme<TPDE>::SolvePDE()
00089 {
00090 for (int j = 0; j <= this->jmax_; j++)
00091 this->V[this->imax_][j] = this->f(j);
00092
00093 for (int i = this->imax_; i > 0; i--)
00094 {
00095 auto pvec = this->A(i, this->V[i]);
00096 auto wvec = this->w(i);
00097 for (int j = 0; j <= this->jmax_; j++)
00098 pvec[j] += wvec[j];
00099 this->V[i - 1] = LUdecomposition(i, pvec);
00100 this->V[i - 1][0] = this->f1(i - 1);
00101 this->V[i - 1][this->jmax_] = this->fu(i - 1);
00102 }
00103
00104 } // namespace pde
00105
00106 #endif // IMPLICITSHEME_H

```

## 7.23 Référence du fichier src/Option.cpp

```
#include "pch.h"
#include "Option.h"
```

Graphe des dépendances par inclusion de Option.cpp:



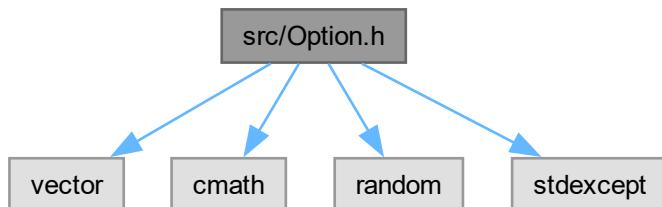
### Espaces de nommage

— namespace [crr](#)

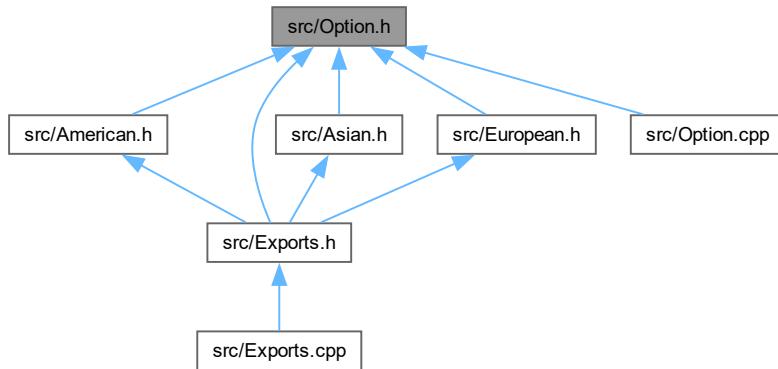
## 7.24 Référence du fichier src/Option.h

```
#include <vector>
#include <cmath>
#include <random>
#include <stdexcept>
```

Graphe des dépendances par inclusion de Option.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Classes

- class [crr::Option](#)

*Classe de base [Option](#) : calcul des paramètres du modèle.*

- struct [crr::Option::HedgingStrategy](#)

*Stratégie de couverture : delta et obligation.*

## Espaces de nommage

- namespace [crr](#)

## 7.25 Option.h

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef OPTION_H
00002 #define OPTION_H
00003
00004 #include <vector>
00005 #include <cmath>
00006 #include <random>
00007 #include <stdexcept>
00008
00009 namespace crr {
00010
00014 class Option {
00015 protected:
00016 double S0_, R_, sigma_, T_;
00017 int N_;
00018 double u_, d_, discount_;
00019
00020 public:
00021 Option(double S0, double R, double sigma, double T, int N);
00030
00035 virtual std::vector<std::vector<double>> treePrice() const = 0;
00036
00041 double price() const;

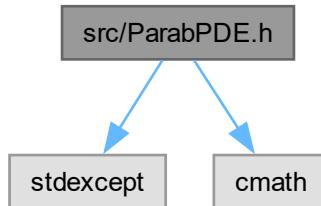
```

```
00042 struct HedgingStrategy {
00043 std::vector<std::vector<double>> delta;
00044 std::vector<std::vector<double>> bond;
00045 };
00046
00047 virtual HedgingStrategy hedgingStrategy() const = 0;
00048
00049 double deltaZero() const;
00050
00051 };
00052
00053 } // namespace crr
00054
00055 #endif // OPTION_H
00056
00057
00058
```

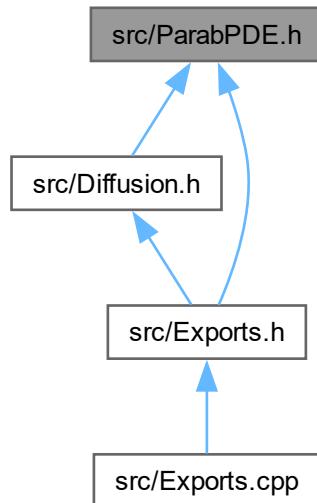
## 7.26 Référence du fichier src/ParabPDE.h

```
#include <stdexcept>
#include <cmath>
```

Graphe des dépendances par inclusion de ParabPDE.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Classes

— class [pde::ParabPDE](#)

*Interface pour EDP paraboliques de type général.*

## Espaces de nommage

## — namespace pde

## 7.27 ParabPDE.h

[Aller à la documentation de ce fichier.](#)

```
00001 #ifndef PARABPDE_H
00002 #define PARABPDE_H
00003
00004 #include <stdexcept>
00005 #include <cmath>
00006
00007 namespace pde {
00008
00012 class ParabPDE {
00013 protected:
00014 double T_, Smin_, Smax_;
00015
00016 public:
00017 ParabPDE(double T, double Smin, double Smax)
00018 : T_(T), Smin_(Smin), Smax_(Smax)
00019 {
00020 if (T_ <= 0.0 || Smin_ >= Smax_ || Smin_ < 0)
00021 throw std::invalid_argument("Domaine EDP invalide");
```

```

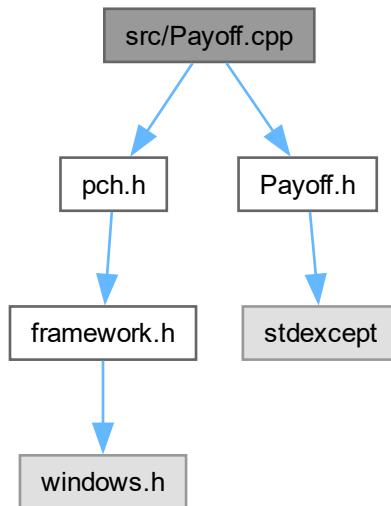
00022 }
00023
00027 virtual double a(double t, double S) const = 0;
00028 virtual double b(double t, double S) const = 0;
00029 virtual double c(double t, double S) const = 0;
00030 virtual double d(double t, double S) const = 0;
00031
00036 virtual double Terminal(double S) const = 0;
00037
00042 virtual double Lower(double t) const = 0;
00043
00048 virtual double Upper(double t) const = 0;
00049
00050 double T() const { return T_; }
00051 double Smin() const { return Smin_; }
00052 double Smax() const { return Smax_; }
00053 };
00054
00055 } // namespace pde
00056
00057 #endif // PARABPDE_H
00058

```

## 7.28 Référence du fichier src/Payoff.cpp

```
#include "pch.h"
#include "Payoff.h"
```

Graphe des dépendances par inclusion de Payoff.cpp:



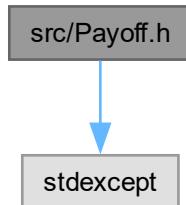
### Espaces de nommage

- namespace opt

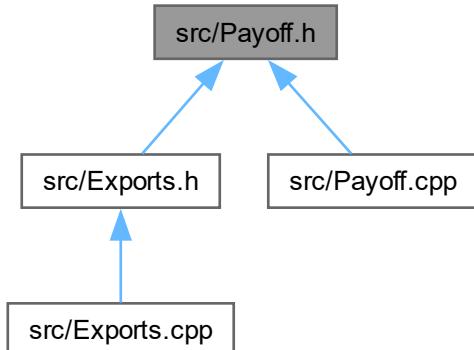
## 7.29 Référence du fichier src/Payoff.h

```
#include <stdexcept>
```

Graphe des dépendances par inclusion de Payoff.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



### Classes

- class [opt::Payoff](#)

*Classe abstraite représentant le payoff d'une option.*

- class [opt::PayoffCall](#)

*Payoff d'une option d'achat européenne (call).*

- class [opt::PayoffPut](#)

*Payoff d'une option de vente européenne (put).*

- class [opt::PayoffDigitCall](#)

*Payoff* d'un call digital.

- class [opt::PayoffDigitPut](#)

*Payoff* d'un put digital.

- class [opt::PayoffDoubleDigit](#)

*Payoff* d'une option double-digital.

- class [opt::PayoffBull](#)

*Payoff* d'une option bull spread.

- class [opt::PayoffBear](#)

*Payoff* d'une option bear spread.

- class [opt::PayoffStrangle](#)

*Payoff* d'une option strangle.

- class [opt::PayoffButterfly](#)

*Payoff* d'une option butterfly.

## Espaces de nommage

- namespace [opt](#)

## 7.30 Payoff.h

[Aller à la documentation de ce fichier.](#)

```

00001 #ifndef PAYOFF_H
00002 #define PAYOFF_H
00003
00004 #include <stdexcept>
00005
00006 namespace opt {
00007
00011 class Payoff {
00012 public:
00018 virtual double operator() (double S) const = 0;
00019 };
00020
00024 class PayoffCall : public Payoff {
00025 private:
00026 double K_;
00027
00028 public:
00032 PayoffCall(double K);
00033
00034 double operator() (double S) const override;
00035 };
00036
00040 class PayoffPut : public Payoff {
00041 private:
00042 double K_;
00043
00044 public:
00045 PayoffPut(double K);
00046 double operator() (double S) const override;

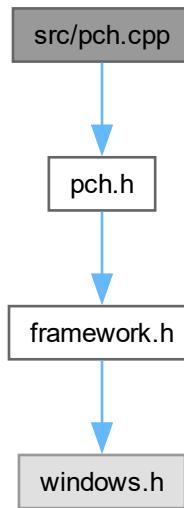
```

```
00047 };
00048
00052 class PayoffDigitCall : public Payoff {
00053 private:
00054 double K_;
00055
00056 public:
00057 PayoffDigitCall(double K);
00058 double operator()(double S) const override;
00059 };
00060
00064 class PayoffDigitPut : public Payoff {
00065 private:
00066 double K_;
00067
00068 public:
00069 PayoffDigitPut(double K);
00070 double operator()(double S) const override;
00071 };
00072
00076 class PayoffDoubleDigit : public Payoff {
00077 private:
00078 double K1_;
00079 double K2_;
00080
00081 public:
00082 PayoffDoubleDigit(double K1, double K2);
00083
00084 double operator()(double S) const override;
00085 };
00086
00094 class PayoffBull : public Payoff {
00095 private:
00096 double K1_;
00097 double K2_;
00098
00099 public:
00100 PayoffBull(double K1, double K2);
00101 double operator()(double S) const override;
00102 };
00103
00107 class PayoffBear : public Payoff {
00108 private:
00109 double K1_;
00110 double K2_;
00111
00112 public:
00113 PayoffBear(double K1, double K2);
00114 double operator()(double S) const override;
00115 };
00116
00120 class PayoffStrangle : public Payoff {
00121 private:
00122 double K1_;
00123 double K2_;
00124
00125 public:
00126 PayoffStrangle(double K1, double K2);
00127 double operator()(double S) const override;
00128 };
00129
00133 class PayoffButterfly : public Payoff {
00134 private:
00135 double K1_;
00136 double K2_;
00137
00138 public:
00139 PayoffButterfly(double K1, double K2);
00140 double operator()(double S) const override;
00141 };
00142
00143 } // namespace opt
00144
00145 #endif // PAYOFF_H
00146
```

## 7.31 Référence du fichier src/pch.cpp

```
#include "pch.h"
```

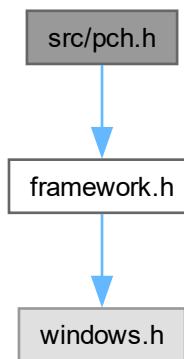
Graphe des dépendances par inclusion de pch.cpp:



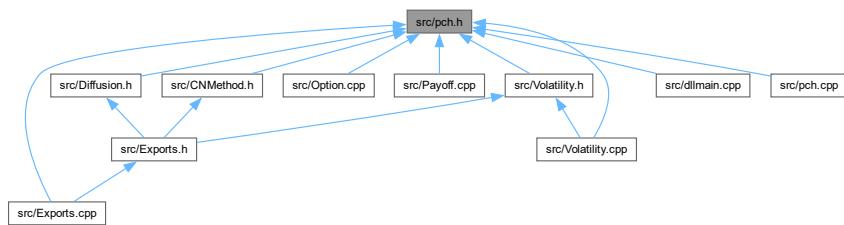
## 7.32 Référence du fichier src/pch.h

```
#include "framework.h"
```

Graphe des dépendances par inclusion de pch.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## 7.33 pch.h

[Aller à la documentation de ce fichier.](#)

```

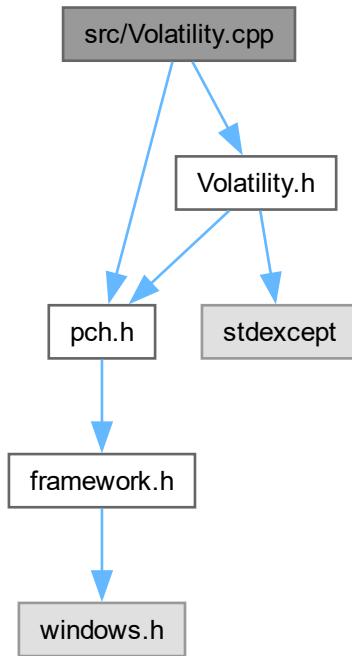
00001 // pch.h: This is a precompiled header file.
00002 // Files listed below are compiled only once, improving build performance for future builds.
00003 // This also affects IntelliSense performance, including code completion and many code browsing
 features.
00004 // However, files listed here are ALL re-compiled if any one of them is updated between builds.
00005 // Do not add files here that you will be updating frequently as this negates the performance
 advantage.
00006
00007 #ifndef PCH_H
00008 #define PCH_H
00009
00010 // add headers that you want to pre-compile here
00011 #include "framework.h"
00012
00013 #endif //PCH_H

```

## 7.34 Référence du fichier src/Volatility.cpp

```
#include "pch.h"
#include "Volatility.h"
```

Graphe des dépendances par inclusion de Volatility.cpp:



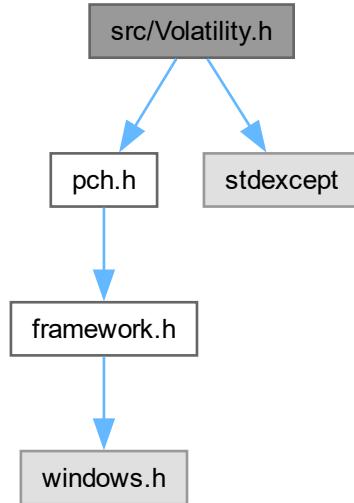
### Espaces de nommage

— namespace [pde](#)

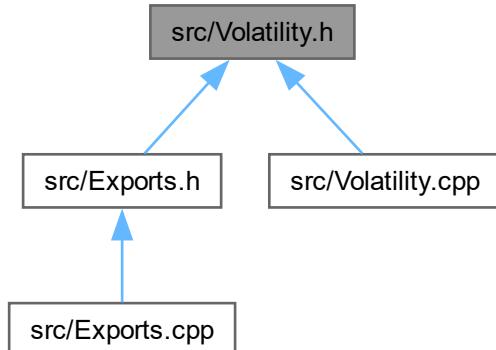
## 7.35 Référence du fichier src/Volatility.h

```
#include "pch.h"
#include <stdexcept>
```

Graphe des dépendances par inclusion de Volatility.h:



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



## Classes

- class [pde::Volatility](#)

*Classe abstraite représentant la volatilité du sous-jacent.*

- class [pde::BSVol](#)

*Volatilité dans le modèle de Black-Scholes.*

— class `pde::LocalVol`

*Volatilité dans un modèle à volatilité locale.*

## Espaces de nommage

— namespace `pde`

## 7.36 Volatility.h

[Aller à la documentation de ce fichier.](#)

```
00001 #include "pch.h"
00002
00003 #ifndef VOLATILITY_H
00004 #define VOLATILITY_H
00005
00006 #include <stdexcept>
00007
00008 namespace pde {
00009
00013 class Volatility {
00014 public:
00021 virtual double operator()(double t, double S) const = 0;
00022 };
00023
00027 class BSVol : public Volatility {
00028 private:
00029 double sigma_;
00030
00031 public:
00035 BSVol(double sigma);
00036
00037 double operator()(double t, double S) const override;
00038 };
00039
00043 class LocalVol : public Volatility {
00044 private:
00045 double alfa_, beta_;
00046
00047 public:
00052 LocalVol(double alfa, double beta);
00053
00054 double operator()(double t, double S) const override;
00055 };
00056
00057 } // namespace pde
00058
00059 #endif // VOLATILITY_H
```



# Index

— declspec  
  Exports.cpp, 145  
  Exports.h, 199

A  
  pde::CNMethod< TPDE >, 37  
  pde::ImplicitScheme< TPDE >, 70

a  
  pde::Diffusion< TPayoff, TVol >, 44  
  pde::FDMMethod< TPDE >, 55  
  pde::ParabPDE, 91

alfa  
  Exports.cpp, 196  
  Exports.h, 205

American  
  crr::American< TPayoff >, 18

Asian  
  crr::Asian< TPayoff, TAggregator >, 27

AsianCallArithmetic  
  Exports.cpp, 150

AsianCallGeometric  
  Exports.cpp, 151

AsianCallLookMax  
  Exports.cpp, 151

AsianPutArithmetic  
  Exports.cpp, 152

AsianPutGeometric  
  Exports.cpp, 153

AsianPutLookMin  
  Exports.cpp, 153

B  
  pde::CNMethod< TPDE >, 37  
  pde::ImplicitScheme< TPDE >, 71

b  
  pde::Diffusion< TPayoff, TVol >, 44  
  pde::FDMMethod< TPDE >, 55  
  pde::ParabPDE, 91

beta  
  Exports.cpp, 196  
  Exports.h, 205

bond  
  crr::Option::HedgingStrategy, 65

BSVol  
  pde::BSVol, 32

C  
  pde::CNMethod< TPDE >, 38  
  pde::ImplicitScheme< TPDE >, 71

c  
  pde::Diffusion< TPayoff, TVol >, 44  
  pde::FDMMethod< TPDE >, 56  
  pde::ParabPDE, 91

CNMethod  
  pde::CNMethod< TPDE >, 36

CNMethod.h  
  CNMETHOD\_H, 126

CNMETHOD\_H  
  CNMethod.h, 126

crr, 9  
  crr::Aggregator, 13  
    operator(), 14

crr::American< opt::PayoffCall >  
  Exports.cpp, 154

crr::American< opt::PayoffPut >  
  Exports.cpp, 154

crr::American< TPayoff >, 14  
  American, 18  
  deltaRR, 19  
  hedgingStrategy, 19  
  priceRR, 19  
  treePrice, 20

crr::Arithmetic, 21  
  operator(), 22

crr::Asian< TPayoff, TAggregator >, 23  
  Asian, 27  
  deltaMC, 28  
  hedgingStrategy, 28  
  priceMC, 28  
  terminalValues, 29  
  treePrice, 29

crr::European< opt::PayoffBear >  
  Exports.cpp, 154

crr::European< opt::PayoffBull >  
  Exports.cpp, 154

crr::European< opt::PayoffButterfly >  
  Exports.cpp, 154

crr::European< opt::PayoffCall >  
  Exports.cpp, 154

crr::European< opt::PayoffDigitCall >  
  Exports.cpp, 155

crr::European< opt::PayoffDigitPut >  
  Exports.cpp, 155

crr::European< opt::PayoffDoubleDigit >  
  Exports.cpp, 155

crr::European< opt::PayoffPut >  
  Exports.cpp, 155

crr::European< opt::PayoffStrangle >  
  Exports.cpp, 155

crr::European< TPayoff >, 46  
 European, 49  
 hedgingStrategy, 49  
 treePrice, 50  
 crr::Geometric, 61  
 operator(), 62  
 crr::LookMax, 78  
 operator(), 79  
 crr::LookMin, 79  
 operator(), 81  
 crr::Option, 81  
 d\_, 87  
 deltaZero, 85  
 discount\_, 87  
 hedgingStrategy, 85  
 N\_, 87  
 Option, 84  
 price, 86  
 R\_, 87  
 S0\_, 87  
 sigma\_, 87  
 T\_, 87  
 treePrice, 86  
 u\_, 88  
 crr::Option::HedgingStrategy, 64  
 bond, 65  
 delta, 65

D  
 pde::CNMethod< TPDE >, 39  
 pde::ImplicitScheme< TPDE >, 72

d  
 pde::Diffusion< TPayoff, TVol >, 44  
 pde::FDMETHOD< TPDE >, 56  
 pde::ParabPDE, 91

d\_  
 crr::Option, 87

del  
 pde::FDMETHOD< TPDE >::Grid, 64

delta  
 crr::Option::HedgingStrategy, 65  
 Exports.cpp, 156  
 pde::FDMETHOD< TPDE >, 56

deltaMC  
 crr::Asian< TPayoff, TAggregator >, 28

deltaRR  
 crr::American< TPayoff >, 19

deltaZero  
 crr::Option, 85

Diffusion  
 pde::Diffusion< TPayoff, TVol >, 44

DiffusionBearBS  
 Exports.cpp, 143

DiffusionBearVL  
 Exports.cpp, 143

DiffusionBullBS  
 Exports.cpp, 143

DiffusionBullVL  
 Exports.cpp, 143

DiffusionButterflyBS  
 Exports.cpp, 144

DiffusionButterflyVL  
 Exports.cpp, 144

DiffusionDigitCallBS  
 Exports.cpp, 144

DiffusionDigitCallVL  
 Exports.cpp, 144

DiffusionDigitPutBS  
 Exports.cpp, 144

DiffusionDigitPutVL  
 Exports.cpp, 144

DiffusionDoubleDigitBS  
 Exports.cpp, 144

DiffusionDoubleDigitVL  
 Exports.cpp, 144

DiffusionPutBS  
 Exports.cpp, 144

DiffusionPutVL  
 Exports.cpp, 144

DiffusionStrangleBS  
 Exports.cpp, 145

DiffusionStrangleVL  
 Exports.cpp, 145

discount\_  
 crr::Option, 87

DllMain  
 dllmain.cpp, 130

dllmain.cpp  
 DllMain, 130

dS\_  
 pde::FDMETHOD< TPDE >, 60

dt\_  
 pde::FDMETHOD< TPDE >, 60

E  
 pde::CNMethod< TPDE >, 39  
 pde::ImplicitScheme< TPDE >, 72

European  
 crr::European< TPayoff >, 49

Exports.cpp  
 \_\_declspec, 145  
 alfa, 196  
 AsianCallArithmetic, 150  
 AsianCallGeometric, 151  
 AsianCallLookMax, 151  
 AsianPutArithmetic, 152  
 AsianPutGeometric, 153  
 AsianPutLookMin, 153  
 beta, 196  
 crr::American< opt::PayoffCall >, 154  
 crr::American< opt::PayoffPut >, 154  
 crr::European< opt::PayoffBear >, 154  
 crr::European< opt::PayoffBull >, 154  
 crr::European< opt::PayoffButterfly >, 154  
 crr::European< opt::PayoffCall >, 154  
 crr::European< opt::PayoffDigitCall >, 155  
 crr::European< opt::PayoffDigitPut >, 155  
 crr::European< opt::PayoffDoubleDigit >, 155

crr::European< opt::PayoffPut >, 155  
crr::European< opt::PayoffStrangle >, 155  
delta, 156  
DiffusionBearBS, 143  
DiffusionBearVL, 143  
DiffusionBullBS, 143  
DiffusionBullVL, 143  
DiffusionButterflyBS, 144  
DiffusionButterflyVL, 144  
DiffusionDigitCallBS, 144  
DiffusionDigitCallVL, 144  
DiffusionDigitPutBS, 144  
DiffusionDigitPutVL, 144  
DiffusionDoubleDigitBS, 144  
DiffusionDoubleDigitVL, 144  
DiffusionPutBS, 144  
DiffusionPutVL, 144  
DiffusionStrangleBS, 145  
DiffusionStrangleVL, 145  
G, 196  
imax, 196  
jmax, 196  
K, 196  
K1, 196  
K2, 196  
makeVariantFromArray, 156  
N, 196  
R, 196  
reportError, 157  
S, 196  
S0, 197  
SAFE\_DOUBLE, 142, 158–172  
SAFE\_VARIANT, 143, 172–192  
sigma, 197  
Smax, 197  
Smin, 197  
SolvePDE, 192  
solver, 192  
T, 197  
t, 197  
toVariant, 194, 195  
Exports.h  
    \_\_declspec, 199  
    alfa, 205  
    beta, 205  
    imax, 205  
    jmax, 205  
    K, 205  
    K1, 205  
    K2, 205  
    N, 205  
    R, 205  
    S, 205  
    sigma, 205  
    Smax, 206  
    Smin, 206  
    T, 206  
F  
    pde::CNMethod< TPDE >, 40  
    pde::ImplicitScheme< TPDE >, 72  
f  
    pde::FDMMethod< TPDE >, 57  
FDMMethod  
    pde::FDMMethod< TPDE >, 55  
fl  
    pde::FDMMethod< TPDE >, 57  
framework.h  
    WIN32\_LEAN\_AND\_MEAN, 220  
fu  
    pde::FDMMethod< TPDE >, 58  
G  
    Exports.cpp, 196  
    pde::CNMethod< TPDE >, 40  
    pde::ImplicitScheme< TPDE >, 73  
grid  
    pde::FDMMethod< TPDE >, 58  
hedgingStrategy  
    crr::American< TPayoff >, 19  
    crr::Asian< TPayoff, TAggregator >, 28  
    crr::European< TPayoff >, 49  
    crr::Option, 85  
imax  
    Exports.cpp, 196  
    Exports.h, 205  
imax\_  
    pde::FDMMethod< TPDE >, 60  
ImplicitScheme  
    pde::ImplicitScheme< TPDE >, 69  
jmax  
    Exports.cpp, 196  
    Exports.h, 205  
jmax\_  
    pde::FDMMethod< TPDE >, 61  
K  
    Exports.cpp, 196  
    Exports.h, 205  
K1  
    Exports.cpp, 196  
    Exports.h, 205  
K2  
    Exports.cpp, 196  
    Exports.h, 205  
LocalVol  
    pde::LocalVol, 77  
Lower  
    pde::Diffusion< TPayoff, TVol >, 44  
    pde::ParabPDE, 92  
LUDecomposition  
    pde::ImplicitScheme< TPDE >, 73  
makeVariantFromArray  
    Exports.cpp, 156

N  
 Exports.cpp, 196  
 Exports.h, 205  
 N\_  
 crr::Option, 87  
 operator()  
     crr::Aggregator, 14  
     crr::Arithmetic, 22  
     crr::Geometric, 62  
     crr::LookMax, 79  
     crr::LookMin, 81  
     opt::Payoff, 95  
     opt::PayoffBear, 97  
     opt::PayoffBull, 99  
     opt::PayoffButterfly, 101  
     opt::PayoffCall, 104  
     opt::PayoffDigitCall, 105  
     opt::PayoffDigitPut, 107  
     opt::PayoffDoubleDigit, 110  
     opt::PayoffPut, 111  
     opt::PayoffStrangle, 113  
     pde::BSVol, 32  
     pde::LocalVol, 77  
     pde::Volatility, 115  
 opt, 10  
 opt::Payoff, 95  
     operator(), 95  
 opt::PayoffBear, 96  
     operator(), 97  
     PayoffBear, 97  
 opt::PayoffBull, 98  
     operator(), 99  
     PayoffBull, 99  
 opt::PayoffButterfly, 100  
     operator(), 101  
     PayoffButterfly, 101  
 opt::PayoffCall, 102  
     operator(), 104  
     PayoffCall, 103  
 opt::PayoffDigitCall, 104  
     operator(), 105  
     PayoffDigitCall, 105  
 opt::PayoffDigitPut, 106  
     operator(), 107  
     PayoffDigitPut, 107  
 opt::PayoffDoubleDigit, 108  
     operator(), 110  
     PayoffDoubleDigit, 109  
 opt::PayoffPut, 110  
     operator(), 111  
     PayoffPut, 111  
 opt::PayoffStrangle, 112  
     operator(), 113  
     PayoffStrangle, 113  
 Option  
     crr::Option, 84  
 ParabPDE  
     pde::ParabPDE, 90  
 PayoffBear  
     opt::PayoffBear, 97  
 PayoffBull  
     opt::PayoffBull, 99  
 PayoffButterfly  
     opt::PayoffButterfly, 101  
 PayoffCall  
     opt::PayoffCall, 103  
 PayoffDigitCall  
     opt::PayoffDigitCall, 105  
 PayoffDigitPut  
     opt::PayoffDigitPut, 107  
 PayoffDoubleDigit  
     opt::PayoffDoubleDigit, 109  
 PayoffPut  
     opt::PayoffPut, 111  
 PayoffStrangle  
     opt::PayoffStrangle, 113  
 pde, 10  
 pde::BSVol, 30  
     BSVol, 32  
     operator(), 32  
 pde::CNMethod< TPDE >, 33  
     A, 37  
     B, 37  
     C, 38  
     CNMethod, 36  
     D, 39  
     E, 39  
     F, 40  
     G, 40  
 pde::Diffusion< TPayoff, TVol >, 41  
     a, 44  
     b, 44  
     c, 44  
     d, 44  
     Diffusion, 44  
     Lower, 44  
     Terminal, 45  
     Upper, 45  
 pde::FDMethod< TPDE >, 51  
     a, 55  
     b, 55  
     c, 56  
     d, 56  
     delta, 56  
     dS\_, 60  
     dt\_, 60  
     f, 57  
     FDMethod, 55  
     fl, 57  
     fu, 58  
     grid, 58  
     imax\_, 60  
     jmax\_, 61  
     pde\_, 61  
     S, 59

t, 59  
V, 61  
v, 59  
pde::FDMMethod< TPDE >::Grid, 63  
del, 64  
val, 64  
pde::ImplicitScheme< TPDE >, 65  
A, 70  
B, 71  
C, 71  
D, 72  
E, 72  
F, 72  
G, 73  
ImplicitScheme, 69  
LUDecomposition, 73  
SolvePDE, 74  
w, 74  
pde::LocalVol, 75  
LocalVol, 77  
operator(), 77  
pde::ParabPDE, 88  
a, 91  
b, 91  
c, 91  
d, 91  
Lower, 92  
ParabPDE, 90  
Smax, 92  
Smax\_, 94  
Smin, 92  
Smin\_, 94  
T, 93  
T\_, 94  
Terminal, 93  
Upper, 93  
pde::Volatility, 114  
operator(), 115  
pde\_  
    pde::FDMMethod< TPDE >, 61  
price  
    crr::Option, 86  
priceMC  
    crr::Asian< TPayoff, TAggregator >, 28  
priceRR  
    crr::American< TPayoff >, 19  
  
R  
    Exports.cpp, 196  
    Exports.h, 205  
R\_  
    crr::Option, 87  
reportError  
    Exports.cpp, 157  
  
S  
    Exports.cpp, 196  
    Exports.h, 205  
    pde::FDMMethod< TPDE >, 59  
  
S0  
    Exports.cpp, 197  
S0\_  
    crr::Option, 87  
SAFE\_DOUBLE  
    Exports.cpp, 142, 158–172  
SAFE\_VARIANT  
    Exports.cpp, 143, 172–192  
sigma  
    Exports.cpp, 197  
    Exports.h, 205  
sigma\_  
    crr::Option, 87  
Smax  
    Exports.cpp, 197  
    Exports.h, 206  
    pde::ParabPDE, 92  
Smax\_  
    pde::ParabPDE, 94  
Smin  
    Exports.cpp, 197  
    Exports.h, 206  
    pde::ParabPDE, 92  
Smin\_  
    pde::ParabPDE, 94  
SolvePDE  
    Exports.cpp, 192  
    pde::ImplicitScheme< TPDE >, 74  
solver  
    Exports.cpp, 192  
src/Aggregator.h, 117, 118  
src/American.h, 119, 120  
src/Asian.h, 122, 123  
src/CNMethod.h, 125, 127  
src/Diffusion.h, 127, 129  
src/dllmain.cpp, 130  
src/European.h, 130, 132  
src/Exports.cpp, 133  
src/Exports.h, 197, 206  
src/FDMMethod.h, 216, 217  
src/framework.h, 219, 220  
src/ImplicitScheme.h, 220, 221  
src/Option.cpp, 222  
src/Option.h, 223, 224  
src/ParabPDE.h, 225, 226  
src/Payoff.cpp, 227  
src/Payoff.h, 228, 229  
src/pch.cpp, 231  
src/pch.h, 231, 232  
src/Volatility.cpp, 232  
src/Volatility.h, 233, 235  
  
T  
    Exports.cpp, 197  
    Exports.h, 206  
    pde::ParabPDE, 93  
t  
    Exports.cpp, 197  
    pde::FDMMethod< TPDE >, 59

T\_

crr::Option, [87](#)  
pde::ParabPDE, [94](#)

Terminal

pde::Diffusion< TPayoff, TVol >, [45](#)  
pde::ParabPDE, [93](#)

terminalValues

crr::Asian< TPayoff, TAggregator >, [29](#)

toVariant

Exports.cpp, [194](#), [195](#)

treePrice

crr::American< TPayoff >, [20](#)  
crr::Asian< TPayoff, TAggregator >, [29](#)  
crr::European< TPayoff >, [50](#)  
crr::Option, [86](#)

U\_

crr::Option, [88](#)

Upper

pde::Diffusion< TPayoff, TVol >, [45](#)  
pde::ParabPDE, [93](#)

V

pde::FDMMethod< TPDE >, [61](#)

v

pde::FDMMethod< TPDE >, [59](#)

val

pde::FDMMethod< TPDE >::Grid, [64](#)

w

pde::ImplicitScheme< TPDE >, [74](#)

WIN32\_LEAN\_AND\_MEAN

framework.h, [220](#)