

# TEB Package

0.1

Generated by Doxygen 1.8.6

Thu Mar 5 2015 14:26:41



# Contents

<b>1</b>	<b>Timed-Elastic-Band Package Documentation</b>	<b>1</b>
1.1	Overview . . . . .	1
1.2	TEB Optimization Problem . . . . .	1
1.2.1	Problem Formulation . . . . .	1
1.2.2	Solution of the TEB Optimization Problem . . . . .	2
1.3	Further Information . . . . .	2
1.3.1	Download . . . . .	2
1.3.2	Installation . . . . .	2
<b>2</b>	<b>Download TEB-Package</b>	<b>3</b>
<b>3</b>	<b>Install TEB-Package</b>	<b>7</b>
<b>4</b>	<b>Solver Overview</b>	<b>9</b>
<b>5</b>	<b>Matlab Interface</b>	<b>11</b>
<b>6</b>	<b>Test List</b>	<b>15</b>
<b>7</b>	<b>Todo List</b>	<b>17</b>
<b>8</b>	<b>Bug List</b>	<b>21</b>
<b>9</b>	<b>Module Index</b>	<b>23</b>
9.1	Modules . . . . .	23
<b>10</b>	<b>Namespace Index</b>	<b>25</b>
10.1	Namespace List . . . . .	25
<b>11</b>	<b>Hierarchical Index</b>	<b>27</b>
11.1	Class Hierarchy . . . . .	27
<b>12</b>	<b>Class Index</b>	<b>29</b>
12.1	Class List . . . . .	29
<b>13</b>	<b>File Index</b>	<b>31</b>

13.1 File List . . . . .	31
<b>14 Module Documentation</b>	<b>33</b>
14.1 Controller . . . . .	33
14.1.1 Detailed Description . . . . .	34
14.1.2 Typedef Documentation . . . . .	34
14.1.2.1 EdgeControlBounds . . . . .	34
14.1.2.2 EdgePositiveTime . . . . .	35
14.1.2.3 EdgeStateBounds . . . . .	35
14.2 Solver . . . . .	36
14.2.1 Detailed Description . . . . .	36
14.3 Simulation . . . . .	37
14.3.1 Detailed Description . . . . .	37
14.4 Visualization . . . . .	38
14.4.1 Detailed Description . . . . .	38
<b>15 Namespace Documentation</b>	<b>39</b>
15.1 teb Namespace Reference . . . . .	39
15.1.1 Typedef Documentation . . . . .	41
15.1.1.1 BackupStackType . . . . .	41
15.1.1.2 EigenScalar . . . . .	41
15.1.1.3 EigenScalarD . . . . .	42
15.1.2 Enumeration Type Documentation . . . . .	42
15.1.2.1 BOUND_TYPE . . . . .	42
15.1.2.2 BOUND_VARS . . . . .	42
15.1.2.3 FiniteDifferences . . . . .	42
15.1.2.4 FUNCT_TYPE . . . . .	42
15.1.2.5 HessianInit . . . . .	42
15.1.2.6 HessianMethod . . . . .	43
15.1.2.7 NloptAlgorithms . . . . .	43
15.1.2.8 NumericalIntegrators . . . . .	43
<b>16 Class Documentation</b>	<b>45</b>
16.1 teb::BaseController Class Reference . . . . .	45
16.1.1 Detailed Description . . . . .	46
16.1.2 Member Function Documentation . . . . .	46
16.1.2.1 firstControl . . . . .	46
16.1.2.2 firstState . . . . .	46
16.1.2.3 getAbsoluteTimeVec . . . . .	46
16.1.2.4 getDt . . . . .	47
16.1.2.5 getN . . . . .	47

16.1.2.6	<a href="#">getStateCtrlInfoMat</a>	47
16.1.2.7	<a href="#">lastState</a>	47
16.1.2.8	<a href="#">resetController</a>	47
16.1.2.9	<a href="#">returnControlInputSequence</a>	47
16.1.2.10	<a href="#">step</a>	48
16.2	<a href="#">teb::BaseEdge&lt; D, FuncType, Vertices &gt; Class Template Reference</a>	49
16.2.1	<a href="#">Detailed Description</a>	51
16.2.2	<a href="#">Member Typedef Documentation</a>	51
16.2.2.1	<a href="#">EdgeContainer</a>	51
16.2.2.2	<a href="#">ValueVector</a>	52
16.2.2.3	<a href="#">ValueVectorMap</a>	52
16.2.3	<a href="#">Constructor &amp; Destructor Documentation</a>	52
16.2.3.1	<a href="#">BaseEdge</a>	52
16.2.3.2	<a href="#">BaseEdge</a>	52
16.2.3.3	<a href="#">~BaseEdge</a>	52
16.2.4	<a href="#">Member Function Documentation</a>	52
16.2.4.1	<a href="#">allocateMemory</a>	52
16.2.4.2	<a href="#">backupJacobian</a>	52
16.2.4.3	<a href="#">calculateVertexDimensions</a>	52
16.2.4.4	<a href="#">computeHessian</a>	53
16.2.4.5	<a href="#">computeJacobian</a>	53
16.2.4.6	<a href="#">computeValues</a>	54
16.2.4.7	<a href="#">dimension</a>	54
16.2.4.8	<a href="#">discardJacobianBackup</a>	54
16.2.4.9	<a href="#">functType</a>	54
16.2.4.10	<a href="#">getCustomData</a>	54
16.2.4.11	<a href="#">getJacobianBackup</a>	54
16.2.4.12	<a href="#">getVertex</a>	55
16.2.4.13	<a href="#">getVertex</a>	55
16.2.4.14	<a href="#">hessians</a>	55
16.2.4.15	<a href="#">hessians</a>	55
16.2.4.16	<a href="#">isBoundConstraint</a>	55
16.2.4.17	<a href="#">jacobians</a>	55
16.2.4.18	<a href="#">jacobians</a>	55
16.2.4.19	<a href="#">noVertices</a>	55
16.2.4.20	<a href="#">restoreJacobian</a>	55
16.2.4.21	<a href="#">values</a>	56
16.2.4.22	<a href="#">values</a>	56
16.2.4.23	<a href="#">valuesData</a>	56
16.2.4.24	<a href="#">valuesData</a>	56

16.2.4.25 valuesMap . . . . .	56
16.2.4.26 vertices . . . . .	56
16.2.5 Member Data Documentation . . . . .	56
16.2.5.1 _hessians . . . . .	56
16.2.5.2 _jacob_backup . . . . .	57
16.2.5.3 _jacobians . . . . .	57
16.2.5.4 _values . . . . .	57
16.2.5.5 _vertices . . . . .	57
16.2.5.6 _vertices_dim . . . . .	57
16.2.5.7 Dimension . . . . .	57
16.2.5.8 NoVertices . . . . .	58
16.3 teb::BaseEdge< D, FuncType > Class Template Reference . . . . .	58
16.3.1 Detailed Description . . . . .	60
16.3.2 Member Typedef Documentation . . . . .	61
16.3.2.1 EdgeContainer . . . . .	61
16.3.2.2 ValueVector . . . . .	61
16.3.2.3 ValueVectorMap . . . . .	61
16.3.3 Constructor & Destructor Documentation . . . . .	61
16.3.3.1 BaseEdge . . . . .	61
16.3.3.2 ~BaseEdge . . . . .	61
16.3.4 Member Function Documentation . . . . .	61
16.3.4.1 allocateMemory . . . . .	61
16.3.4.2 backupJacobian . . . . .	62
16.3.4.3 calculateVertexDimensions . . . . .	62
16.3.4.4 computeHessian . . . . .	62
16.3.4.5 computeJacobian . . . . .	62
16.3.4.6 computeValues . . . . .	63
16.3.4.7 dimension . . . . .	63
16.3.4.8 discardJacobianBackup . . . . .	63
16.3.4.9 functType . . . . .	63
16.3.4.10 getCustomData . . . . .	63
16.3.4.11 getJacobianBackup . . . . .	63
16.3.4.12 getVertex . . . . .	64
16.3.4.13 getVertex . . . . .	64
16.3.4.14 hessians . . . . .	64
16.3.4.15 hessians . . . . .	64
16.3.4.16 isBoundConstraint . . . . .	64
16.3.4.17 jacobians . . . . .	64
16.3.4.18 jacobians . . . . .	64
16.3.4.19 noVertices . . . . .	64

16.3.4.20	<a href="#">resizeVertexContainer</a>	65
16.3.4.21	<a href="#">restoreJacobian</a>	65
16.3.4.22	<a href="#">setVertex</a>	65
16.3.4.23	<a href="#">values</a>	65
16.3.4.24	<a href="#">values</a>	65
16.3.4.25	<a href="#">valuesData</a>	65
16.3.4.26	<a href="#">valuesData</a>	65
16.3.4.27	<a href="#">valuesMap</a>	66
16.3.4.28	<a href="#">vertices</a>	66
16.3.4.29	<a href="#">vertices</a>	66
16.3.5	<a href="#">Member Data Documentation</a>	66
16.3.5.1	<a href="#">_hessians</a>	66
16.3.5.2	<a href="#">_jacob_backup</a>	66
16.3.5.3	<a href="#">_jacobians</a>	66
16.3.5.4	<a href="#">_values</a>	67
16.3.5.5	<a href="#">_vertices</a>	67
16.3.5.6	<a href="#">_vertices_dim</a>	67
16.3.5.7	<a href="#">Dimension</a>	67
16.4	<a href="#">teb::BaseSolver Class Reference</a>	67
16.4.1	<a href="#">Detailed Description</a>	69
16.4.2	<a href="#">Member Typedef Documentation</a>	69
16.4.2.1	<a href="#">EdgeContainer</a>	69
16.4.2.2	<a href="#">VertexContainer</a>	69
16.4.3	<a href="#">Constructor &amp; Destructor Documentation</a>	69
16.4.3.1	<a href="#">BaseSolver</a>	69
16.4.3.2	<a href="#">~BaseSolver</a>	70
16.4.4	<a href="#">Member Function Documentation</a>	70
16.4.4.1	<a href="#">applyIncrement</a>	70
16.4.4.2	<a href="#">applyOptVec</a>	70
16.4.4.3	<a href="#">backupVertices</a>	70
16.4.4.4	<a href="#">discardBackupVertices</a>	70
16.4.4.5	<a href="#">getDimEqualities</a>	71
16.4.4.6	<a href="#">getDimInequalities</a>	71
16.4.4.7	<a href="#">getDimObjectives</a>	71
16.4.4.8	<a href="#">getOptVecCopy</a>	71
16.4.4.9	<a href="#">getOptVecDimension</a>	71
16.4.4.10	<a href="#">initWorkspaces</a>	72
16.4.4.11	<a href="#">restoreVertices</a>	72
16.4.4.12	<a href="#">restoreVerticesButKeepBackup</a>	72
16.4.4.13	<a href="#">setConfig</a>	72

16.4.4.14	<code>solve</code>	72
16.4.4.15	<code>solveImpl</code>	73
16.4.5	Member Data Documentation	73
16.4.5.1	<code>_active_vertices</code>	73
16.4.5.2	<code>_equalities</code>	73
16.4.5.3	<code>_equalities_dim</code>	73
16.4.5.4	<code>_graph_structure_modified</code>	74
16.4.5.5	<code>_inequalities</code>	74
16.4.5.6	<code>_inequalities_dim</code>	74
16.4.5.7	<code>_no_vert_backups</code>	74
16.4.5.8	<code>_objective_dim</code>	74
16.4.5.9	<code>_objectives</code>	74
16.4.5.10	<code>_opt_vec_dim</code>	75
16.4.5.11	<code>cfg</code>	75
16.4.5.12	<code>EIGEN_MAKE_ALIGNED_OPERATOR_NEW</code>	75
16.5	<code>teb::BaseSolverLeastSquares</code> Class Reference	75
16.5.1	Detailed Description	78
16.5.2	Member Typedef Documentation	78
16.5.2.1	<code>EdgeContainer</code>	78
16.5.2.2	<code>VertexContainer</code>	78
16.5.3	Constructor & Destructor Documentation	78
16.5.3.1	<code>BaseSolverLeastSquares</code>	78
16.5.4	Member Function Documentation	78
16.5.4.1	<code>adaptWeights</code>	78
16.5.4.2	<code>applyIncrement</code>	79
16.5.4.3	<code>applyOptVec</code>	79
16.5.4.4	<code>backupVertices</code>	79
16.5.4.5	<code>buildValueVector</code>	80
16.5.4.6	<code>discardBackupVertices</code>	80
16.5.4.7	<code>getChi2</code>	81
16.5.4.8	<code>getDimEqualities</code>	81
16.5.4.9	<code>getDimInequalities</code>	81
16.5.4.10	<code>getDimObjectives</code>	81
16.5.4.11	<code>getOptVecCopy</code>	81
16.5.4.12	<code>getOptVecDimension</code>	82
16.5.4.13	<code>getValueDimension</code>	82
16.5.4.14	<code>initWorkspaces</code>	82
16.5.4.15	<code>restoreVertices</code>	82
16.5.4.16	<code>restoreVerticesButKeepBackup</code>	83
16.5.4.17	<code>setConfig</code>	83



16.5.4.18 solve	83
16.5.4.19 solveImpl	83
16.5.5 Member Data Documentation	84
16.5.5.1 _active_vertices	84
16.5.5.2 _equalities	84
16.5.5.3 _equalities_dim	84
16.5.5.4 _graph_structure_modified	84
16.5.5.5 _inequalities	84
16.5.5.6 _inequalities_dim	85
16.5.5.7 _no_vert_backups	85
16.5.5.8 _objective_dim	85
16.5.5.9 _objectives	85
16.5.5.10 _opt_vec_dim	85
16.5.5.11 _val_dim	86
16.5.5.12 _values	86
16.5.5.13 _weight_adapt_count	86
16.5.5.14 _weight_equalities	86
16.5.5.15 _weight_inequalities	86
16.5.5.16 cfg	86
16.5.5.17 EIGEN_MAKE_ALIGNED_OPERATOR_NEW	86
16.6 teb::BaseSolverNonlinearProgramDense Class Reference	87
16.6.1 Detailed Description	89
16.6.2 Member Typedef Documentation	90
16.6.2.1 EdgeContainer	90
16.6.2.2 MatMapRowMajor	90
16.6.2.3 VertexContainer	90
16.6.3 Constructor & Destructor Documentation	90
16.6.3.1 BaseSolverNonlinearProgramDense	90
16.6.4 Member Function Documentation	90
16.6.4.1 applyIncrement	90
16.6.4.2 applyOptVec	90
16.6.4.3 backupVertices	91
16.6.4.4 buildEqualityConstraintJacobian	91
16.6.4.5 buildEqualityConstraintValueVector	91
16.6.4.6 buildInequalityConstraintJacobian	91
16.6.4.7 buildInequalityConstraintValueVector	91
16.6.4.8 buildObjectiveGradient	91
16.6.4.9 buildObjectiveValue	92
16.6.4.10 calculateLagrangianGradient	92
16.6.4.11 calculateLagrangianHessian	92

16.6.4.12	<a href="#">calculateLagrangianHessianFullBFGS</a>	92
16.6.4.13	<a href="#">calculateLagrangianHessianNumerically</a>	92
16.6.4.14	<a href="#">discardBackupVertices</a>	93
16.6.4.15	<a href="#">getDimEqualities</a>	93
16.6.4.16	<a href="#">getDimInequalities</a>	93
16.6.4.17	<a href="#">getDimObjectives</a>	93
16.6.4.18	<a href="#">getOptVecCopy</a>	93
16.6.4.19	<a href="#">getOptVecDimension</a>	94
16.6.4.20	<a href="#">initHessianBFGS</a>	94
16.6.4.21	<a href="#">initializeLagrangeMultiplier</a>	94
16.6.4.22	<a href="#">initSolverWorkspace</a>	94
16.6.4.23	<a href="#">initWorkspaces</a>	94
16.6.4.24	<a href="#">restoreVertices</a>	94
16.6.4.25	<a href="#">restoreVerticesButKeepBackup</a>	95
16.6.4.26	<a href="#">setConfig</a>	95
16.6.4.27	<a href="#">solve</a>	95
16.6.4.28	<a href="#">solveImpl</a>	96
16.6.5	<a href="#">Member Data Documentation</a>	97
16.6.5.1	<a href="#">_active_vertices</a>	97
16.6.5.2	<a href="#">_equalities</a>	97
16.6.5.3	<a href="#">_equalities_dim</a>	97
16.6.5.4	<a href="#">_equality_jacobian</a>	97
16.6.5.5	<a href="#">_equality_values</a>	97
16.6.5.6	<a href="#">_graph_structure_modified</a>	98
16.6.5.7	<a href="#">_inequalities</a>	98
16.6.5.8	<a href="#">_inequalities_dim</a>	98
16.6.5.9	<a href="#">_inequality_jacobian</a>	98
16.6.5.10	<a href="#">_inequality_values</a>	98
16.6.5.11	<a href="#">_lagrangian_gradient</a>	98
16.6.5.12	<a href="#">_lagrangian_gradient_backup</a>	98
16.6.5.13	<a href="#">_lagrangian_hessian</a>	99
16.6.5.14	<a href="#">_multiplier_eq</a>	99
16.6.5.15	<a href="#">_multiplier_ineq</a>	99
16.6.5.16	<a href="#">_no_vert_backups</a>	99
16.6.5.17	<a href="#">_objective_dim</a>	99
16.6.5.18	<a href="#">_objective_gradient</a>	99
16.6.5.19	<a href="#">_objective_value</a>	99
16.6.5.20	<a href="#">_objectives</a>	99
16.6.5.21	<a href="#">_opt_vec_dim</a>	100
16.6.5.22	<a href="#">cfg</a>	100

16.6.5.23 EIGEN_MAKE_ALIGNED_OPERATOR_NEW . . . . .	100
16.7 teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index > Class Template Reference . . .	100
16.7.1 Detailed Description . . . . .	103
16.7.2 Member Typedef Documentation . . . . .	103
16.7.2.1 EdgeContainer . . . . .	103
16.7.2.2 ValueVector . . . . .	103
16.7.2.3 ValueVectorMap . . . . .	104
16.7.3 Constructor & Destructor Documentation . . . . .	104
16.7.3.1 BoundConstraint . . . . .	104
16.7.4 Member Function Documentation . . . . .	104
16.7.4.1 allocateMemory . . . . .	104
16.7.4.2 backupJacobian . . . . .	104
16.7.4.3 calculateVertexDimensions . . . . .	104
16.7.4.4 computeHessian . . . . .	105
16.7.4.5 computeJacobian . . . . .	105
16.7.4.6 computeValues . . . . .	105
16.7.4.7 dimension . . . . .	105
16.7.4.8 discardJacobianBackup . . . . .	105
16.7.4.9 functType . . . . .	105
16.7.4.10 getBounds . . . . .	105
16.7.4.11 getCustomData . . . . .	106
16.7.4.12 getJacobianBackup . . . . .	106
16.7.4.13 getVertex . . . . .	106
16.7.4.14 getVertex . . . . .	107
16.7.4.15 hessians . . . . .	107
16.7.4.16 hessians . . . . .	107
16.7.4.17 isBoundConstraint . . . . .	107
16.7.4.18 jacobians . . . . .	107
16.7.4.19 jacobians . . . . .	107
16.7.4.20 noVertices . . . . .	107
16.7.4.21 restoreJacobian . . . . .	107
16.7.4.22 setBounds . . . . .	108
16.7.4.23 setBounds . . . . .	108
16.7.4.24 setBounds . . . . .	108
16.7.4.25 setBounds . . . . .	108
16.7.4.26 setBounds . . . . .	109
16.7.4.27 setBounds . . . . .	109
16.7.4.28 values . . . . .	109
16.7.4.29 values . . . . .	110
16.7.4.30 valuesData . . . . .	110

16.7.4.31	valuesData	110
16.7.4.32	valuesMap	110
16.7.4.33	vertices	110
16.7.5	Member Data Documentation	110
16.7.5.1	_bounds	110
16.7.5.2	_hessians	110
16.7.5.3	_jacob_backup	111
16.7.5.4	_jacobians	111
16.7.5.5	_values	111
16.7.5.6	_vertices	111
16.7.5.7	_vertices_dim	111
16.7.5.8	BoundType	111
16.7.5.9	Dimension	112
16.7.5.10	NoBounds	112
16.7.5.11	NoVertices	112
16.8	teb::Config Class Reference	112
16.8.1	Detailed Description	113
16.8.2	Member Data Documentation	113
16.8.2.1	optim	113
16.8.2.2	teb	113
16.8.2.3	util	113
16.9	teb::ControlVertex< q > Class Template Reference	113
16.9.1	Detailed Description	116
16.9.2	Member Typedef Documentation	116
16.9.2.1	BackupStackType	116
16.9.2.2	ControlVector	116
16.9.2.3	FixedCtrls	116
16.9.2.4	ScalarType	117
16.9.2.5	VertexContainer	117
16.9.3	Constructor & Destructor Documentation	117
16.9.3.1	ControlVertex	117
16.9.3.2	ControlVertex	117
16.9.3.3	ControlVertex	117
16.9.4	Member Function Documentation	117
16.9.4.1	controls	117
16.9.4.2	controls	117
16.9.4.3	dimension	118
16.9.4.4	dimensionFree	118
16.9.4.5	discardTop	118
16.9.4.6	fixed_ctrls	118

16.9.4.7	<a href="#">getData</a>	118
16.9.4.8	<a href="#">getDataFree</a>	118
16.9.4.9	<a href="#">getOptVecIdx</a>	118
16.9.4.10	<a href="#">isFixedAll</a>	119
16.9.4.11	<a href="#">isFixedAny</a>	119
16.9.4.12	<a href="#">isFixedComp</a>	119
16.9.4.13	<a href="#">operator=</a>	119
16.9.4.14	<a href="#">plus</a>	119
16.9.4.15	<a href="#">plus</a>	120
16.9.4.16	<a href="#">plusFree</a>	120
16.9.4.17	<a href="#">pop</a>	120
16.9.4.18	<a href="#">push</a>	120
16.9.4.19	<a href="#">setControls</a>	120
16.9.4.20	<a href="#">setControls</a>	120
16.9.4.21	<a href="#">setFixedAll</a>	120
16.9.4.22	<a href="#">setFixedCtrl</a>	121
16.9.4.23	<a href="#">setFixedCtrls</a>	121
16.9.4.24	<a href="#">setFixedCtrls</a>	121
16.9.4.25	<a href="#">setFixedCtrls</a>	121
16.9.4.26	<a href="#">setFree</a>	121
16.9.4.27	<a href="#">setOptVecIdx</a>	122
16.9.4.28	<a href="#">stackSize</a>	123
16.9.4.29	<a href="#">top</a>	123
16.9.5	<a href="#">Friends And Related Function Documentation</a>	123
16.9.5.1	<a href="#">operator!=</a>	123
16.9.5.2	<a href="#">operator&lt;&lt;</a>	123
16.9.5.3	<a href="#">operator==</a>	123
16.9.6	<a href="#">Member Data Documentation</a>	123
16.9.6.1	<a href="#">_backup</a>	123
16.9.6.2	<a href="#">_controls</a>	124
16.9.6.3	<a href="#">_dimension</a>	124
16.9.6.4	<a href="#">_fixed_ctrls</a>	124
16.9.6.5	<a href="#">_opt_vec_idx</a>	124
16.9.6.6	<a href="#">DimControls</a>	124
16.9.6.7	<a href="#">Dimension</a>	124
16.10	<a href="#">teb::CustomBoundData Struct Reference</a>	124
16.10.1	<a href="#">Detailed Description</a>	125
16.10.2	<a href="#">Member Data Documentation</a>	125
16.10.2.1	<a href="#">EIGEN_MAKE_ALIGNED_OPERATOR_NEW</a>	125
16.10.2.2	<a href="#">index</a>	125

16.10.2.3 lower . . . . .	125
16.10.2.4 no_lower . . . . .	125
16.10.2.5 no_upper . . . . .	125
16.10.2.6 upper . . . . .	125
16.11 teb::EdgeMinimizeTime Class Reference . . . . .	126
16.11.1 Detailed Description . . . . .	128
16.11.2 Member Typedef Documentation . . . . .	128
16.11.2.1 EdgeContainer . . . . .	128
16.11.2.2 ValueVector . . . . .	128
16.11.2.3 ValueVectorMap . . . . .	128
16.11.3 Constructor & Destructor Documentation . . . . .	129
16.11.3.1 EdgeMinimizeTime . . . . .	129
16.11.4 Member Function Documentation . . . . .	129
16.11.4.1 allocateMemory . . . . .	129
16.11.4.2 backupJacobian . . . . .	129
16.11.4.3 calculateVertexDimensions . . . . .	129
16.11.4.4 computeHessian . . . . .	129
16.11.4.5 computeJacobian . . . . .	130
16.11.4.6 computeValues . . . . .	130
16.11.4.7 dimension . . . . .	130
16.11.4.8 discardJacobianBackup . . . . .	130
16.11.4.9 functType . . . . .	130
16.11.4.10 getCustomData . . . . .	130
16.11.4.11 getJacobianBackup . . . . .	130
16.11.4.12 getVertex . . . . .	130
16.11.4.13 getVertex . . . . .	131
16.11.4.14 Hessians . . . . .	131
16.11.4.15 Hessians . . . . .	131
16.11.4.16 isBoundConstraint . . . . .	131
16.11.4.17 Jacobians . . . . .	131
16.11.4.18 Jacobians . . . . .	131
16.11.4.19 noVertices . . . . .	131
16.11.4.20 restoreJacobian . . . . .	131
16.11.4.21 setData . . . . .	132
16.11.4.22 values . . . . .	132
16.11.4.23 values . . . . .	132
16.11.4.24 valuesData . . . . .	132
16.11.4.25 valuesData . . . . .	132
16.11.4.26 valuesMap . . . . .	132
16.11.4.27 vertices . . . . .	132

16.11.5 Member Data Documentation . . . . .	133
16.11.5.1 _data . . . . .	133
16.11.5.2 _hessians . . . . .	133
16.11.5.3 _jacob_backup . . . . .	133
16.11.5.4 _jacobians . . . . .	133
16.11.5.5 _values . . . . .	133
16.11.5.6 _vertices . . . . .	133
16.11.5.7 _vertices_dim . . . . .	133
16.11.5.8 Dimension . . . . .	133
16.11.5.9 NoVertices . . . . .	134
16.12teb::EdgeQuadraticForm< p, q > Class Template Reference . . . . .	134
16.12.1 Detailed Description . . . . .	136
16.12.2 Member Typedef Documentation . . . . .	137
16.12.2.1 EdgeContainer . . . . .	137
16.12.2.2 StateVector . . . . .	137
16.12.2.3 ValueVector . . . . .	137
16.12.2.4 ValueVectorMap . . . . .	137
16.12.3 Constructor & Destructor Documentation . . . . .	137
16.12.3.1 EdgeQuadraticForm . . . . .	137
16.12.4 Member Function Documentation . . . . .	138
16.12.4.1 allocateMemory . . . . .	138
16.12.4.2 backupJacobian . . . . .	138
16.12.4.3 calculateVertexDimensions . . . . .	138
16.12.4.4 computeHessian . . . . .	138
16.12.4.5 computeJacobian . . . . .	139
16.12.4.6 computeValues . . . . .	139
16.12.4.7 dimension . . . . .	139
16.12.4.8 discardJacobianBackup . . . . .	139
16.12.4.9 functType . . . . .	139
16.12.4.10getCustomData . . . . .	139
16.12.4.11getJacobianBackup . . . . .	140
16.12.4.12getVertex . . . . .	140
16.12.4.13getVertex . . . . .	140
16.12.4.14hessians . . . . .	140
16.12.4.15hessians . . . . .	140
16.12.4.16sBoundConstraint . . . . .	140
16.12.4.17jacobians . . . . .	140
16.12.4.18jacobians . . . . .	140
16.12.4.19noVertices . . . . .	141
16.12.4.20restoreJacobian . . . . .	141

16.12.4.21	setReference	141
16.12.4.22	setWeights	141
16.12.4.23	setWeights	141
16.12.4.24	setWeights	141
16.12.4.25	values	141
16.12.4.26	values	141
16.12.4.27	valuesData	142
16.12.4.28	valuesData	142
16.12.4.29	valuesMap	142
16.12.4.30	vertices	142
16.12.5	Member Data Documentation	142
16.12.5.1	_hessians	142
16.12.5.2	_jacob_backup	142
16.12.5.3	_jacobians	142
16.12.5.4	_Q	143
16.12.5.5	_R	143
16.12.5.6	_ref_state	143
16.12.5.7	_values	143
16.12.5.8	_vertices	143
16.12.5.9	_vertices_dim	143
16.12.5.10	Dimension	143
16.12.5.11	NoVertices	143
16.13	teb::EdgeSystemDynamics< p, q, central > Class Template Reference	144
16.13.1	Detailed Description	146
16.13.2	Member Typedef Documentation	147
16.13.2.1	EdgeContainer	147
16.13.2.2	ValueVector	147
16.13.2.3	ValueVectorMap	147
16.13.3	Constructor & Destructor Documentation	147
16.13.3.1	EdgeSystemDynamics	147
16.13.4	Member Function Documentation	147
16.13.4.1	allocateMemory	147
16.13.4.2	backupJacobian	148
16.13.4.3	calculateVertexDimensions	148
16.13.4.4	computeHessian	148
16.13.4.5	computeJacobian	149
16.13.4.6	computeValues	149
16.13.4.7	dimension	149
16.13.4.8	discardJacobianBackup	149
16.13.4.9	functType	149



16.13.4.10	getCustomData	149
16.13.4.11	getJacobianBackup	150
16.13.4.12	getVertex	150
16.13.4.13	getVertex	150
16.13.4.14	hessians	150
16.13.4.15	hessians	150
16.13.4.16	sBoundConstraint	150
16.13.4.17	jacobians	150
16.13.4.18	jacobians	150
16.13.4.19	noVertices	151
16.13.4.20	restoreJacobian	151
16.13.4.21	setSystemDynamics	151
16.13.4.22	values	151
16.13.4.23	values	151
16.13.4.24	valuesData	151
16.13.4.25	valuesData	151
16.13.4.26	valuesMap	151
16.13.4.27	vertices	152
16.13.5	Friends And Related Function Documentation	152
16.13.5.1	SystemDynamics	152
16.13.6	Member Data Documentation	152
16.13.6.1	_hessians	152
16.13.6.2	_jacob_backup	152
16.13.6.3	_jacobians	152
16.13.6.4	_system	152
16.13.6.5	_values	152
16.13.6.6	_vertices	153
16.13.6.7	_vertices_dim	153
16.13.6.8	Dimension	153
16.13.6.9	NoVertices	153
16.14	teb::EdgeType Class Reference	153
16.14.1	Detailed Description	155
16.14.2	Member Typedef Documentation	155
16.14.2.1	EdgeContainer	155
16.14.2.2	ValueVectorMap	155
16.14.3	Constructor & Destructor Documentation	155
16.14.3.1	EdgeType	155
16.14.3.2	~EdgeType	155
16.14.4	Member Function Documentation	156
16.14.4.1	allocateMemory	156

16.14.4.2 backupJacobian . . . . .	156
16.14.4.3 computeHessian . . . . .	156
16.14.4.4 computeJacobian . . . . .	156
16.14.4.5 computeValues . . . . .	156
16.14.4.6 dimension . . . . .	156
16.14.4.7 discardJacobianBackup . . . . .	156
16.14.4.8 functType . . . . .	157
16.14.4.9 getCustomData . . . . .	157
16.14.4.10getJacobianBackup . . . . .	157
16.14.4.11getVertex . . . . .	157
16.14.4.12getVertex . . . . .	157
16.14.4.13hessians . . . . .	157
16.14.4.14hessians . . . . .	157
16.14.4.15sBoundConstraint . . . . .	157
16.14.4.16jacobians . . . . .	157
16.14.4.17jacobians . . . . .	158
16.14.4.18noVertices . . . . .	158
16.14.4.19restoreJacobian . . . . .	158
16.14.4.20valuesData . . . . .	158
16.14.4.21valuesData . . . . .	158
16.14.4.22valuesMap . . . . .	158
16.14.5 Member Data Documentation . . . . .	158
16.14.5.1 _hessians . . . . .	158
16.14.5.2 _jacob_backup . . . . .	159
16.14.5.3 _jacobians . . . . .	159
16.15teb::ExplicitEuler< p, q > Class Template Reference . . . . .	159
16.15.1 Detailed Description . . . . .	160
16.15.2 Member Typedef Documentation . . . . .	160
16.15.2.1 ControlVector . . . . .	160
16.15.2.2 StateVector . . . . .	160
16.15.3 Constructor & Destructor Documentation . . . . .	160
16.15.3.1 ExplicitEuler . . . . .	160
16.15.3.2 ~ExplicitEuler . . . . .	160
16.15.4 Member Function Documentation . . . . .	160
16.15.4.1 integrate . . . . .	161
16.16teb::Config::Optim::Solver::NonlinearProgram::Hessian Struct Reference . . . . .	161
16.16.1 Detailed Description . . . . .	161
16.16.2 Member Data Documentation . . . . .	162
16.16.2.1 bfgs_damped_mode . . . . .	162
16.16.2.2 hessian_init . . . . .	162

16.16.2.3 hessian_init_identity_scale . . . . .	162
16.16.2.4 hessian_method . . . . .	162
16.17teb::HessianWorkspace Class Reference . . . . .	162
16.17.1 Detailed Description . . . . .	163
16.17.2 Member Typedef Documentation . . . . .	164
16.17.2.1 WorkspaceMatrix . . . . .	164
16.17.3 Constructor & Destructor Documentation . . . . .	164
16.17.3.1 HessianWorkspace . . . . .	164
16.17.3.2 ~HessianWorkspace . . . . .	164
16.17.4 Member Function Documentation . . . . .	164
16.17.4.1 allocate . . . . .	164
16.17.4.2 getWorkspace . . . . .	164
16.17.5 Member Data Documentation . . . . .	165
16.17.5.1 _workspace . . . . .	165
16.18teb::HyperGraph Class Reference . . . . .	165
16.18.1 Detailed Description . . . . .	166
16.18.2 Member Typedef Documentation . . . . .	167
16.18.2.1 EdgeContainer . . . . .	167
16.18.2.2 VertexContainer . . . . .	167
16.18.3 Member Function Documentation . . . . .	167
16.18.3.1 activeVertices . . . . .	167
16.18.3.2 activeVertices . . . . .	167
16.18.3.3 addActiveVertex . . . . .	167
16.18.3.4 addEdgeEquality . . . . .	167
16.18.3.5 addEdgeInequality . . . . .	168
16.18.3.6 addEdgeObjective . . . . .	169
16.18.3.7 clearActiveVertices . . . . .	169
16.18.3.8 clearEdges . . . . .	169
16.18.3.9 clearGraph . . . . .	169
16.18.3.10equalities . . . . .	169
16.18.3.11equalities . . . . .	170
16.18.3.12nequalities . . . . .	170
16.18.3.13nequalities . . . . .	170
16.18.3.14sGraphModified . . . . .	170
16.18.3.15notifyGraphModified . . . . .	170
16.18.3.16objectives . . . . .	171
16.18.3.17objectives . . . . .	171
16.18.4 Member Data Documentation . . . . .	171
16.18.4.1 _active_vertices . . . . .	171
16.18.4.2 _constraints_eq . . . . .	171

16.18.4.3 <code>_constraints_ineq</code> . . . . .	171
16.18.4.4 <code>_graph_modified</code> . . . . .	171
16.18.4.5 <code>_objectives</code> . . . . .	171
16.19teb::JacobianWorkspace Class Reference . . . . .	172
16.19.1 Detailed Description . . . . .	172
16.19.2 Member Typedef Documentation . . . . .	173
16.19.2.1 <code>WorkspaceMatrix</code> . . . . .	173
16.19.3 Constructor & Destructor Documentation . . . . .	173
16.19.3.1 <code>JacobianWorkspace</code> . . . . .	173
16.19.3.2 <code>~JacobianWorkspace</code> . . . . .	173
16.19.4 Member Function Documentation . . . . .	173
16.19.4.1 <code>allocate</code> . . . . .	173
16.19.4.2 <code>getWorkspace</code> . . . . .	173
16.19.5 Member Data Documentation . . . . .	174
16.19.5.1 <code>_workspace</code> . . . . .	174
16.20teb::Config::Optim::Solver::NonlinearProgram::LineSearch Struct Reference . . . . .	174
16.20.1 Detailed Description . . . . .	174
16.20.2 Member Data Documentation . . . . .	174
16.20.2.1 <code>alpha_init</code> . . . . .	174
16.20.2.2 <code>beta</code> . . . . .	175
16.20.2.3 <code>sigma</code> . . . . .	175
16.21teb::Config::Optim::Solver::Lsq Struct Reference . . . . .	175
16.21.1 Detailed Description . . . . .	175
16.21.2 Member Data Documentation . . . . .	175
16.21.2.1 <code>soft_constr_epsilon</code> . . . . .	175
16.21.2.2 <code>weight_adaptation_factor</code> . . . . .	176
16.21.2.3 <code>weight_equalities</code> . . . . .	176
16.21.2.4 <code>weight_inequalities</code> . . . . .	176
16.22teb::Config::Optim::Solver::Nlopt Struct Reference . . . . .	176
16.22.1 Detailed Description . . . . .	177
16.22.2 Member Data Documentation . . . . .	177
16.22.2.1 <code>algorithm</code> . . . . .	177
16.22.2.2 <code>max_optimization_time</code> . . . . .	177
16.22.2.3 <code>stopping_criteria_ftol_abs</code> . . . . .	177
16.22.2.4 <code>stopping_criteria_ftol_rel</code> . . . . .	177
16.22.2.5 <code>stopping_criteria_xtol_abs</code> . . . . .	177
16.22.2.6 <code>stopping_criteria_xtol_rel</code> . . . . .	177
16.22.2.7 <code>tolerance_equalities</code> . . . . .	177
16.22.2.8 <code>tolerance_inequalities</code> . . . . .	177
16.23teb::Config::Optim::Solver::NonlinearProgram Struct Reference . . . . .	177

16.23.1 Detailed Description . . . . .	178
16.23.2 Member Data Documentation . . . . .	178
16.23.2.1 hessian . . . . .	178
16.23.2.2 linesearch . . . . .	178
16.24teb::NumericalIntegrator< p, q > Class Template Reference . . . . .	178
16.24.1 Detailed Description . . . . .	179
16.24.2 Member Typedef Documentation . . . . .	180
16.24.2.1 ControlVector . . . . .	180
16.24.2.2 StateVector . . . . .	180
16.24.3 Constructor & Destructor Documentation . . . . .	180
16.24.3.1 NumericalIntegrator . . . . .	180
16.24.3.2 ~NumericalIntegrator . . . . .	180
16.24.4 Member Function Documentation . . . . .	180
16.24.4.1 integrate . . . . .	180
16.25teb::Config::Optim Struct Reference . . . . .	181
16.25.1 Detailed Description . . . . .	181
16.25.2 Member Data Documentation . . . . .	181
16.25.2.1 force_rebuild_optim_graph . . . . .	181
16.25.2.2 solver . . . . .	181
16.26teb::PlotOptions Struct Reference . . . . .	181
16.26.1 Detailed Description . . . . .	182
16.26.2 Member Data Documentation . . . . .	182
16.26.2.1 legend . . . . .	182
16.26.2.2 legend_entries . . . . .	182
16.26.2.3 skip_last_value_right_column . . . . .	182
16.26.2.4 title . . . . .	183
16.26.2.5 ylabels . . . . .	183
16.27teb::RungeKutta5thOrder< p, q > Class Template Reference . . . . .	183
16.27.1 Detailed Description . . . . .	184
16.27.2 Member Typedef Documentation . . . . .	184
16.27.2.1 ControlVector . . . . .	184
16.27.2.2 StateVector . . . . .	184
16.27.3 Constructor & Destructor Documentation . . . . .	184
16.27.3.1 RungeKutta5thOrder . . . . .	184
16.27.3.2 ~RungeKutta5thOrder . . . . .	184
16.27.4 Member Function Documentation . . . . .	184
16.27.4.1 integrate . . . . .	185
16.28teb::RungeKuttaClassic< p, q > Class Template Reference . . . . .	185
16.28.1 Detailed Description . . . . .	186
16.28.2 Member Typedef Documentation . . . . .	186

16.28.2.1 ControlVector . . . . .	186
16.28.2.2 StateVector . . . . .	186
16.28.3 Constructor & Destructor Documentation . . . . .	186
16.28.3.1 RungeKuttaClassic . . . . .	186
16.28.3.2 ~RungeKuttaClassic . . . . .	187
16.28.4 Member Function Documentation . . . . .	187
16.28.4.1 integrate . . . . .	187
16.29teb::SimResults Class Reference . . . . .	187
16.29.1 Detailed Description . . . . .	188
16.29.2 Member Function Documentation . . . . .	188
16.29.2.1 allocateMemory . . . . .	188
16.29.2.2 allocateMemory . . . . .	189
16.29.2.3 conservativeResize . . . . .	189
16.29.2.4 conservativeResize . . . . .	189
16.29.3 Friends And Related Function Documentation . . . . .	189
16.29.3.1 Simulator . . . . .	189
16.29.4 Member Data Documentation . . . . .	189
16.29.4.1 series . . . . .	190
16.30teb::Simulator< p, q > Class Template Reference . . . . .	190
16.30.1 Detailed Description . . . . .	192
16.30.2 Member Typedef Documentation . . . . .	192
16.30.2.1 ControlVector . . . . .	192
16.30.2.2 IntegratorPtr . . . . .	192
16.30.2.3 StateVector . . . . .	193
16.30.3 Constructor & Destructor Documentation . . . . .	193
16.30.3.1 Simulator . . . . .	193
16.30.3.2 Simulator . . . . .	193
16.30.3.3 ~Simulator . . . . .	193
16.30.4 Member Function Documentation . . . . .	193
16.30.4.1 plotResults . . . . .	193
16.30.4.2 sampleTime . . . . .	193
16.30.4.3 saturateControl . . . . .	194
16.30.4.4 setControlInputSaturation . . . . .	194
16.30.4.5 setIntegrator . . . . .	194
16.30.4.6 setIntegrator . . . . .	194
16.30.4.7 setPlotter . . . . .	194
16.30.4.8 setPostStepCallback . . . . .	195
16.30.4.9 setPreSimCallback . . . . .	195
16.30.4.10setPreStepCallback . . . . .	195
16.30.4.11setSampleTime . . . . .	196

16.30.4.12	<code>simClosedLoop</code>	196
16.30.4.13	<code>simClosedLoop</code>	197
16.30.4.14	<code>simOpenAndClosedLoop</code>	197
16.30.4.15	<code>simOpenLoop</code>	198
16.30.4.16	<code>simOpenLoop</code>	198
16.30.4.17	<code>systemStep</code>	199
16.30.5	Friends And Related Function Documentation	199
16.30.5.1	<code>SimResults</code>	199
16.30.5.2	<code>SystemDynamics</code>	199
16.30.6	Member Data Documentation	199
16.30.6.1	<code>_callback_sim_pre</code>	199
16.30.6.2	<code>_callback_step_post</code>	199
16.30.6.3	<code>_callback_step_pre</code>	199
16.30.6.4	<code>_control_bounds</code>	199
16.30.6.5	<code>_controller</code>	199
16.30.6.6	<code>_integrator</code>	200
16.30.6.7	<code>_plotter</code>	200
16.30.6.8	<code>_sample_time</code>	200
16.30.6.9	<code>_system</code>	200
16.31	<code>teb::Config::Optim::Solver</code> Struct Reference	200
16.31.1	Detailed Description	201
16.31.2	Member Data Documentation	201
16.31.2.1	<code>lsq</code>	201
16.31.2.2	<code>nlopt</code>	201
16.31.2.3	<code>nonlin_prog</code>	201
16.31.2.4	<code>solver_iter</code>	201
16.32	<code>teb::SolverLevenbergMarquardtEigenDense</code> Class Reference	201
16.32.1	Detailed Description	204
16.32.2	Member Typedef Documentation	204
16.32.2.1	<code>EdgeContainer</code>	204
16.32.2.2	<code>VertexContainer</code>	204
16.32.3	Constructor & Destructor Documentation	204
16.32.3.1	<code>SolverLevenbergMarquardtEigenDense</code>	204
16.32.4	Member Function Documentation	204
16.32.4.1	<code>adaptWeights</code>	204
16.32.4.2	<code>applyIncrement</code>	205
16.32.4.3	<code>applyOptVec</code>	205
16.32.4.4	<code>backupVertices</code>	205
16.32.4.5	<code>buildJacobian</code>	206
16.32.4.6	<code>buildValueVector</code>	206

16.32.4.7	discardBackupVertices	207
16.32.4.8	getChi2	207
16.32.4.9	getDimEqualities	207
16.32.4.10	getDimInequalities	207
16.32.4.11	getDimObjectives	208
16.32.4.12	getOptVecCopy	208
16.32.4.13	getOptVecDimension	208
16.32.4.14	getValueDimension	208
16.32.4.15	nitWorkspaces	209
16.32.4.16	restoreVertices	209
16.32.4.17	restoreVerticesButKeepBackup	209
16.32.4.18	setConfig	209
16.32.4.19	solve	210
16.32.4.20	solveImpl	210
16.32.5	Member Data Documentation	210
16.32.5.1	_active_vertices	210
16.32.5.2	_equalities	210
16.32.5.3	_equalities_dim	211
16.32.5.4	_graph_structure_modified	211
16.32.5.5	_inequalities	211
16.32.5.6	_inequalities_dim	211
16.32.5.7	_jacobian	211
16.32.5.8	_no_vert_backups	212
16.32.5.9	_objective_dim	212
16.32.5.10	_objectives	212
16.32.5.11	_opt_vec_dim	212
16.32.5.12	_val_dim	212
16.32.5.13	_values	212
16.32.5.14	_weight_adapt_count	212
16.32.5.15	_weight_equalities	213
16.32.5.16	_weight_inequalities	213
16.32.5.17	cfg	213
16.32.5.18	EIGEN_MAKE_ALIGNED_OPERATOR_NEW	213
16.33	teb::SolverLevenbergMarquardtEigenSparse Class Reference	213
16.33.1	Detailed Description	216
16.33.2	Member Typedef Documentation	216
16.33.2.1	EdgeContainer	216
16.33.2.2	VertexContainer	217
16.33.3	Constructor & Destructor Documentation	217
16.33.3.1	SolverLevenbergMarquardtEigenSparse	217



16.33.4 Member Function Documentation	217
16.33.4.1 adaptWeights	217
16.33.4.2 allocateSparseJacobian	217
16.33.4.3 applyIncrement	217
16.33.4.4 applyOptVec	218
16.33.4.5 backupVertices	218
16.33.4.6 buildJacobian	218
16.33.4.7 buildValueVector	219
16.33.4.8 countJacobianColNNZ	219
16.33.4.9 discardBackupVertices	220
16.33.4.10 getChi2	220
16.33.4.11 getDimEqualities	220
16.33.4.12 getDimInequalities	220
16.33.4.13 getDimObjectives	220
16.33.4.14 getOptVecCopy	220
16.33.4.15 getOptVecDimension	221
16.33.4.16 getValueDimension	221
16.33.4.17 nitWorkspaces	221
16.33.4.18 restoreVertices	222
16.33.4.19 restoreVerticesButKeepBackup	222
16.33.4.20 setConfig	222
16.33.4.21 solve	222
16.33.4.22 solveImpl	223
16.33.5 Member Data Documentation	223
16.33.5.1 _active_vertices	223
16.33.5.2 _equalities	223
16.33.5.3 _equalities_dim	223
16.33.5.4 _graph_structure_modified	223
16.33.5.5 _inequalities	224
16.33.5.6 _inequalities_dim	224
16.33.5.7 _jacobian	224
16.33.5.8 _nnz_per_col	224
16.33.5.9 _no_vert_backups	224
16.33.5.10 objective_dim	224
16.33.5.11 objectives	224
16.33.5.12 opt_vec_dim	225
16.33.5.13 sparse_solver	225
16.33.5.14 val_dim	225
16.33.5.15 values	225
16.33.5.16 weight_adapt_count	225

16.33.5.17_weight_equalities . . . . .	225
16.33.5.18_weight_inequalities . . . . .	226
16.33.5.19_cfg . . . . .	226
16.33.5.20_EIGEN_MAKE_ALIGNED_OPERATOR_NEW . . . . .	226
16.34_teb::SolverNloptPackage Class Reference . . . . .	226
16.34.1 Detailed Description . . . . .	229
16.34.2 Member Typedef Documentation . . . . .	229
16.34.2.1 EdgeContainer . . . . .	229
16.34.2.2 VertexContainer . . . . .	229
16.34.3 Constructor & Destructor Documentation . . . . .	229
16.34.3.1 SolverNloptPackage . . . . .	229
16.34.4 Member Function Documentation . . . . .	229
16.34.4.1 applyIncrement . . . . .	229
16.34.4.2 applyOptVec . . . . .	230
16.34.4.3 backupVertices . . . . .	230
16.34.4.4 constraintsEq . . . . .	230
16.34.4.5 constraintsInEq . . . . .	230
16.34.4.6 discardBackupVertices . . . . .	230
16.34.4.7 equalityConstraintFunction . . . . .	231
16.34.4.8 getBoundConstrDimFromStorage . . . . .	231
16.34.4.9 getDimEqualities . . . . .	231
16.34.4.10 getDimInequalities . . . . .	231
16.34.4.11 getDimObjectives . . . . .	231
16.34.4.12 getEqConstrDimFromStorage . . . . .	231
16.34.4.13 getInEqConstrDimFromStorage . . . . .	231
16.34.4.14 getOptVecCopy . . . . .	231
16.34.4.15 getOptVecDimension . . . . .	232
16.34.4.16 getOptVecDimFromStorage . . . . .	232
16.34.4.17 inequalityConstraintFunction . . . . .	232
16.34.4.18 nitWorkspaces . . . . .	232
16.34.4.19 objectiveFunction . . . . .	232
16.34.4.20 objectives . . . . .	232
16.34.4.21 restoreVertices . . . . .	232
16.34.4.22 restoreVerticesButKeepBackup . . . . .	233
16.34.4.23 separateInequalitiesAndBounds . . . . .	233
16.34.4.24 setConfig . . . . .	233
16.34.4.25 solve . . . . .	233
16.34.4.26 solveImpl . . . . .	234
16.34.5 Member Data Documentation . . . . .	234
16.34.5.1 _active_vertices . . . . .	234

16.34.5.2 _bound_constraints . . . . .	234
16.34.5.3 _bound_constraints_dim . . . . .	234
16.34.5.4 _ctol_eq . . . . .	234
16.34.5.5 _ctol_ineq . . . . .	234
16.34.5.6 _equalities . . . . .	234
16.34.5.7 _equalities_dim . . . . .	234
16.34.5.8 _graph_structure_modified . . . . .	235
16.34.5.9 _inequalities . . . . .	235
16.34.5.10_inequalities_dim . . . . .	235
16.34.5.11_inequalities_without_bounds . . . . .	235
16.34.5.12_inequalities_without_bounds_dim . . . . .	235
16.34.5.13_lower_bounds . . . . .	235
16.34.5.14_no_vert_backups . . . . .	235
16.34.5.15_objective_dim . . . . .	236
16.34.5.16_objectives . . . . .	236
16.34.5.17_opt_vec_dim . . . . .	236
16.34.5.18_upper_bounds . . . . .	236
16.34.5.19cfg . . . . .	236
16.34.5.20EIGEN_MAKE_ALIGNED_OPERATOR_NEW . . . . .	236
16.35teb::SolverSQPDense Class Reference . . . . .	237
16.35.1 Detailed Description . . . . .	240
16.35.2 Member Typedef Documentation . . . . .	240
16.35.2.1 EdgeContainer . . . . .	240
16.35.2.2 MatMapRowMajor . . . . .	240
16.35.2.3 VertexContainer . . . . .	240
16.35.3 Constructor & Destructor Documentation . . . . .	240
16.35.3.1 SolverSQPDense . . . . .	240
16.35.3.2 SolverSQPDense . . . . .	240
16.35.4 Member Function Documentation . . . . .	241
16.35.4.1 applyIncrement . . . . .	241
16.35.4.2 applyOptVec . . . . .	242
16.35.4.3 backupVertices . . . . .	242
16.35.4.4 buildEqualityConstraintJacobian . . . . .	242
16.35.4.5 buildEqualityConstraintValueVector . . . . .	242
16.35.4.6 buildInequalityConstraintJacobian . . . . .	243
16.35.4.7 buildInequalityConstraintValueVector . . . . .	243
16.35.4.8 buildObjectiveGradient . . . . .	243
16.35.4.9 buildObjectiveValue . . . . .	243
16.35.4.10calculateLagrangianGradient . . . . .	243
16.35.4.11calculateLagrangianHessian . . . . .	243

16.35.4.12	calculateLagrangianHessianFullBFGS	244
16.35.4.13	calculateLagrangianHessianNumerically	244
16.35.4.14	calculateMerit	244
16.35.4.15	calculateMerit	244
16.35.4.16	calculateMeritDerivative	244
16.35.4.17	checkConvergence	245
16.35.4.18	discardBackupVertices	245
16.35.4.19	getDimEqualities	245
16.35.4.20	getDimInequalities	245
16.35.4.21	getDimObjectives	245
16.35.4.22	getOptVecCopy	245
16.35.4.23	getOptVecDimension	246
16.35.4.24	initHessianBFGS	246
16.35.4.25	initializeLagrangeMultiplier	246
16.35.4.26	initSolverWorkspace	246
16.35.4.27	initSolverWorkspace	247
16.35.4.28	initWorkspaces	247
16.35.4.29	restoreVertices	247
16.35.4.30	restoreVerticesButKeepBackup	247
16.35.4.31	setConfig	247
16.35.4.32	solve	248
16.35.4.33	solveImpl	248
16.35.4.34	solveImpl	249
16.35.5	Member Data Documentation	250
16.35.5.1	_A	250
16.35.5.2	_active_vertices	250
16.35.5.3	_delta	250
16.35.5.4	_dmultiplier_eq	250
16.35.5.5	_dmultiplier_ineq	251
16.35.5.6	_equalities	251
16.35.5.7	_equalities_dim	251
16.35.5.8	_equality_jacobian	251
16.35.5.9	_equality_values	251
16.35.5.10	_graph_structure_modified	251
16.35.5.11	_increment	252
16.35.5.12	_inequalities	252
16.35.5.13	_inequalities_dim	252
16.35.5.14	_inequality_jacobian	252
16.35.5.15	_inequality_values	252
16.35.5.16	_lagrangian_gradient	252

16.35.5.17_lagrangian_gradient_backup . . . . .	253
16.35.5.18_lagrangian_hessian . . . . .	253
16.35.5.19_lb . . . . .	253
16.35.5.20_merit_alpha . . . . .	253
16.35.5.21_merit_grad . . . . .	253
16.35.5.22_multiplier_eq . . . . .	253
16.35.5.23_multiplier_ineq . . . . .	253
16.35.5.24_no_vert_backups . . . . .	253
16.35.5.25_objective_dim . . . . .	254
16.35.5.26_objective_gradient . . . . .	254
16.35.5.27_objective_value . . . . .	254
16.35.5.28_objectives . . . . .	254
16.35.5.29_opt_vec_dim . . . . .	254
16.35.5.30_qdual . . . . .	254
16.35.5.31_qsolver . . . . .	254
16.35.5.32_ub . . . . .	255
16.35.5.33_cfg . . . . .	255
16.35.5.34_EIGEN_MAKE_ALIGNED_OPERATOR_NEW . . . . .	255
16.36teb::StateVertex< p > Class Template Reference . . . . .	255
16.36.1 Detailed Description . . . . .	258
16.36.2 Member Typedef Documentation . . . . .	258
16.36.2.1 BackupStackType . . . . .	258
16.36.2.2 FixedStates . . . . .	258
16.36.2.3 ScalarType . . . . .	258
16.36.2.4 StateVector . . . . .	258
16.36.2.5 VertexContainer . . . . .	258
16.36.3 Constructor & Destructor Documentation . . . . .	259
16.36.3.1 StateVertex . . . . .	259
16.36.3.2 StateVertex . . . . .	259
16.36.3.3 StateVertex . . . . .	259
16.36.3.4 StateVertex . . . . .	259
16.36.4 Member Function Documentation . . . . .	259
16.36.4.1 dimension . . . . .	259
16.36.4.2 dimensionFree . . . . .	259
16.36.4.3 discardTop . . . . .	259
16.36.4.4 fixed_states . . . . .	260
16.36.4.5 getData . . . . .	260
16.36.4.6 getDataFree . . . . .	260
16.36.4.7 getOptVecIdx . . . . .	260
16.36.4.8 isFixedAll . . . . .	260

16.36.4.9 isFixedAny . . . . .	260
16.36.4.10 isFixedComp . . . . .	261
16.36.4.11 operator= . . . . .	261
16.36.4.12 plus . . . . .	261
16.36.4.13 plus . . . . .	261
16.36.4.14 plusFree . . . . .	261
16.36.4.15 pop . . . . .	261
16.36.4.16 push . . . . .	262
16.36.4.17 setFixedAll . . . . .	262
16.36.4.18 setFixedState . . . . .	262
16.36.4.19 setFixedStates . . . . .	262
16.36.4.20 setFixedStates . . . . .	262
16.36.4.21 setFixedStates . . . . .	262
16.36.4.22 setFree . . . . .	263
16.36.4.23 setOptVecIdx . . . . .	263
16.36.4.24 setStates . . . . .	263
16.36.4.25 setStates . . . . .	263
16.36.4.26 stackSize . . . . .	263
16.36.4.27 states . . . . .	263
16.36.4.28 states . . . . .	264
16.36.4.29 top . . . . .	264
16.36.5 Friends And Related Function Documentation . . . . .	264
16.36.5.1 operator!= . . . . .	264
16.36.5.2 operator<< . . . . .	264
16.36.5.3 operator== . . . . .	264
16.36.6 Member Data Documentation . . . . .	265
16.36.6.1 _backup . . . . .	265
16.36.6.2 _dimension . . . . .	265
16.36.6.3 _fixed_states . . . . .	265
16.36.6.4 _opt_vec_idx . . . . .	265
16.36.6.5 _states . . . . .	265
16.36.6.6 Dimension . . . . .	265
16.36.6.7 DimStates . . . . .	265
16.37 teb::SystemDynamics< p, q > Class Template Reference . . . . .	265
16.37.1 Detailed Description . . . . .	266
16.37.2 Member Typedef Documentation . . . . .	267
16.37.2.1 ControlVector . . . . .	267
16.37.2.2 StateVector . . . . .	267
16.37.3 Constructor & Destructor Documentation . . . . .	267
16.37.3.1 SystemDynamics . . . . .	267

16.37.3.2 <code>~SystemDynamics</code> . . . . .	267
16.37.4 Member Function Documentation . . . . .	267
16.37.4.1 <code>finiteElemCentralDiff</code> . . . . .	267
16.37.4.2 <code>finiteElemFwdEuler</code> . . . . .	268
16.37.4.3 <code>stateSpaceModel</code> . . . . .	268
16.37.5 Friends And Related Function Documentation . . . . .	269
16.37.5.1 <code>EdgeSystemDynamics</code> . . . . .	269
16.37.5.2 <code>Simulator</code> . . . . .	269
16.37.6 Member Data Documentation . . . . .	269
16.37.6.1 <code>DimControls</code> . . . . .	269
16.37.6.2 <code>DimStates</code> . . . . .	269
16.38 <code>teb::Config::Teb</code> Struct Reference . . . . .	269
16.38.1 Detailed Description . . . . .	270
16.38.2 Member Data Documentation . . . . .	270
16.38.2.1 <code>diff_method</code> . . . . .	270
16.38.2.2 <code>dt_hyst</code> . . . . .	270
16.38.2.3 <code>dt_min</code> . . . . .	270
16.38.2.4 <code>dt_ref</code> . . . . .	270
16.38.2.5 <code>goal_dist_force_reinit</code> . . . . .	270
16.38.2.6 <code>n_max</code> . . . . .	270
16.38.2.7 <code>n_min</code> . . . . .	270
16.38.2.8 <code>n_pre</code> . . . . .	271
16.38.2.9 <code>teb_iter</code> . . . . .	271
16.39 <code>teb::TebController&lt; p, q &gt;</code> Class Template Reference . . . . .	271
16.39.1 Detailed Description . . . . .	275
16.39.2 Member Typedef Documentation . . . . .	276
16.39.2.1 <code>ControlSequence</code> . . . . .	276
16.39.2.2 <code>ControlVector</code> . . . . .	276
16.39.2.3 <code>EdgeContainer</code> . . . . .	276
16.39.2.4 <code>FixedStates</code> . . . . .	276
16.39.2.5 <code>StateSequence</code> . . . . .	276
16.39.2.6 <code>StateVector</code> . . . . .	277
16.39.2.7 <code>VertexContainer</code> . . . . .	277
16.39.3 Constructor & Destructor Documentation . . . . .	277
16.39.3.1 <code>TebController</code> . . . . .	277
16.39.3.2 <code>TebController</code> . . . . .	277
16.39.3.3 <code>~TebController</code> . . . . .	277
16.39.4 Member Function Documentation . . . . .	277
16.39.4.1 <code>activateControlBounds</code> . . . . .	277
16.39.4.2 <code>activateControlBounds</code> . . . . .	278

16.39.4.3 activateObjectiveQuadraticForm . . . . .	278
16.39.4.4 activateObjectiveTimeOptimal . . . . .	279
16.39.4.5 activateStateBounds . . . . .	279
16.39.4.6 activateStateBounds . . . . .	280
16.39.4.7 buildOptimizationGraph . . . . .	280
16.39.4.8 controlSequence . . . . .	281
16.39.4.9 customOptimizationGraph . . . . .	281
16.39.4.10 customOptimizationGraphHotStart . . . . .	282
16.39.4.11 dt . . . . .	282
16.39.4.12 firstControl . . . . .	282
16.39.4.13 firstControlRef . . . . .	282
16.39.4.14 firstControlSaturated . . . . .	282
16.39.4.15 firstState . . . . .	283
16.39.4.16 firstStateRef . . . . .	283
16.39.4.17 getAbsoluteTimeVec . . . . .	283
16.39.4.18 getActiveVertices . . . . .	283
16.39.4.19 getDt . . . . .	284
16.39.4.20 getM . . . . .	284
16.39.4.21 getN . . . . .	284
16.39.4.22 getSampleOptVecIdxVMat . . . . .	284
16.39.4.23 getStateCtrlInfoMat . . . . .	285
16.39.4.24 getTEBFixedMap . . . . .	285
16.39.4.25 graph . . . . .	286
16.39.4.26 nitOptimization . . . . .	286
16.39.4.27 nitTrajectory . . . . .	286
16.39.4.28 insertStateControlInputPair . . . . .	286
16.39.4.29 lastState . . . . .	287
16.39.4.30 lastStateRef . . . . .	287
16.39.4.31 optimizeTEB . . . . .	287
16.39.4.32 predictControl . . . . .	288
16.39.4.33 pushBackStateControlInputPair . . . . .	288
16.39.4.34 pushFrontStateControlInputPair . . . . .	288
16.39.4.35 removeStateControlInputPair . . . . .	288
16.39.4.36 resampleTrajectory . . . . .	289
16.39.4.37 resetController . . . . .	289
16.39.4.38 resizeTrajectory . . . . .	289
16.39.4.39 returnControlInputSequence . . . . .	290
16.39.4.40 setFixedDt . . . . .	290
16.39.4.41 setFixedGoal . . . . .	290
16.39.4.42 setFixedGoal . . . . .	291



16.39.4.43	setFixedGoal	291
16.39.4.44	setFixedStart	291
16.39.4.45	setFixedStart	291
16.39.4.46	setGoalStatesFixedOrUnfixed	292
16.39.4.47	setGoalStatesFixedOrUnfixed	293
16.39.4.48	setGoalStatesFixedOrUnfixed	293
16.39.4.49	setSolver	293
16.39.4.50	setSystemDynamics	294
16.39.4.51	setupHorizon	294
16.39.4.52	stateSequence	295
16.39.4.53	step	295
16.39.4.54	step	295
16.39.4.55	updateGoal	295
16.39.4.56	updateStart	296
16.39.5	Member Data Documentation	296
16.39.5.1	_active_control_bounds	296
16.39.5.2	_active_quadratic_form	297
16.39.5.3	_active_state_bounds	297
16.39.5.4	_active_system_dynamics	297
16.39.5.5	_active_time_optimal	298
16.39.5.6	_cfg_owned	298
16.39.5.7	_ctrl_seq	298
16.39.5.8	_dt	298
16.39.5.9	_dt_ref	298
16.39.5.10	_fixed_goal_states	298
16.39.5.11	_goal_backup	298
16.39.5.12	_graph	298
16.39.5.13	_no_samples	299
16.39.5.14	_optimized	299
16.39.5.15	_solver	299
16.39.5.16	_state_seq	299
16.39.5.17	cfg	299
16.39.5.18	NoControls	299
16.39.5.19	NoStates	299
16.40	teb::TebPlotter Class Reference	300
16.40.1	Detailed Description	301
16.40.2	Member Enumeration Documentation	301
16.40.2.1	FileFormat	301
16.40.3	Constructor & Destructor Documentation	302
16.40.3.1	TebPlotter	302

16.40.3.2 ~TebPlotter . . . . .	302
16.40.4 Member Function Documentation . . . . .	302
16.40.4.1 clearWindow . . . . .	302
16.40.4.2 closeWindow . . . . .	302
16.40.4.3 completePdfExport . . . . .	302
16.40.4.4 isExportToFileEnabled . . . . .	302
16.40.4.5 plot . . . . .	303
16.40.4.6 plot1DVector . . . . .	303
16.40.4.7 plotCustomKey . . . . .	304
16.40.4.8 plotMulti . . . . .	304
16.40.4.9 plotMulti . . . . .	304
16.40.4.10plotTEB . . . . .	304
16.40.4.11plotTEB . . . . .	305
16.40.4.12plotTwoCol . . . . .	305
16.40.4.13plotTwoCol . . . . .	305
16.40.4.14setOutputToFile . . . . .	306
16.40.4.15setOutputToWindow . . . . .	306
16.40.4.16spyMatrix . . . . .	306
16.40.4.17switchWindow . . . . .	306
16.40.5 Member Data Documentation . . . . .	307
16.40.5.1 _file_export . . . . .	307
16.40.5.2 _pdf_flag . . . . .	307
16.40.5.3 pipe . . . . .	307
16.41 teb::TimeDiff Class Reference . . . . .	307
16.41.1 Detailed Description . . . . .	310
16.41.2 Member Typedef Documentation . . . . .	310
16.41.2.1 VertexContainer . . . . .	310
16.41.3 Constructor & Destructor Documentation . . . . .	310
16.41.3.1 TimeDiff . . . . .	310
16.41.3.2 TimeDiff . . . . .	310
16.41.3.3 TimeDiff . . . . .	310
16.41.3.4 TimeDiff . . . . .	310
16.41.3.5 ~TimeDiff . . . . .	311
16.41.4 Member Function Documentation . . . . .	311
16.41.4.1 dimension . . . . .	311
16.41.4.2 dimensionFree . . . . .	311
16.41.4.3 discardTop . . . . .	311
16.41.4.4 dt . . . . .	311
16.41.4.5 dt . . . . .	311
16.41.4.6 getData . . . . .	311

16.41.4.7	<a href="#">getDataFree</a>	312
16.41.4.8	<a href="#">getOptVecIdx</a>	312
16.41.4.9	<a href="#">isFixedAll</a>	312
16.41.4.10	<a href="#">isFixedAny</a>	312
16.41.4.11	<a href="#">isFixedComp</a>	312
16.41.4.12	<a href="#">operator()</a>	312
16.41.4.13	<a href="#">operator=</a>	313
16.41.4.14	<a href="#">plus</a>	313
16.41.4.15	<a href="#">plus</a>	313
16.41.4.16	<a href="#">plusFree</a>	313
16.41.4.17	<a href="#">pop</a>	313
16.41.4.18	<a href="#">push</a>	313
16.41.4.19	<a href="#">setFixedAll</a>	313
16.41.4.20	<a href="#">setFree</a>	314
16.41.4.21	<a href="#">setOptVecIdx</a>	314
16.41.4.22	<a href="#">stackSize</a>	314
16.41.4.23	<a href="#">top</a>	314
16.41.5	<a href="#">Friends And Related Function Documentation</a>	314
16.41.5.1	<a href="#">operator&lt;&lt;</a>	314
16.41.6	<a href="#">Member Data Documentation</a>	314
16.41.6.1	<a href="#">_backup</a>	314
16.41.6.2	<a href="#">_dt</a>	315
16.41.6.3	<a href="#">_fixed</a>	315
16.41.6.4	<a href="#">_opt_vec_idx</a>	315
16.41.6.5	<a href="#">Dimension</a>	315
16.42	<a href="#">teb::SimResults::TimeSeries Struct Reference</a>	315
16.42.1	<a href="#">Detailed Description</a>	315
16.42.2	<a href="#">Member Data Documentation</a>	316
16.42.2.1	<a href="#">controls</a>	316
16.42.2.2	<a href="#">dt</a>	316
16.42.2.3	<a href="#">states</a>	316
16.42.2.4	<a href="#">time</a>	316
16.43	<a href="#">teb::Config::Utilities Struct Reference</a>	316
16.43.1	<a href="#">Detailed Description</a>	316
16.44	<a href="#">teb::VertexType Class Reference</a>	316
16.44.1	<a href="#">Detailed Description</a>	318
16.44.2	<a href="#">Member Typedef Documentation</a>	318
16.44.2.1	<a href="#">VertexContainer</a>	318
16.44.3	<a href="#">Member Function Documentation</a>	318
16.44.3.1	<a href="#">dimension</a>	318

16.44.3.2 dimensionFree . . . . .	318
16.44.3.3 discardTop . . . . .	319
16.44.3.4 getData . . . . .	319
16.44.3.5 getDataFree . . . . .	319
16.44.3.6 getOptVecIdx . . . . .	319
16.44.3.7 isFixedAll . . . . .	319
16.44.3.8 isFixedAny . . . . .	319
16.44.3.9 isFixedComp . . . . .	319
16.44.3.10plus . . . . .	320
16.44.3.11plus . . . . .	320
16.44.3.12plusFree . . . . .	320
16.44.3.13pop . . . . .	320
16.44.3.14push . . . . .	320
16.44.3.15setFree . . . . .	320
16.44.3.16setOptVecIdx . . . . .	320
16.44.3.17stackSize . . . . .	320
16.44.3.18top . . . . .	321
16.44.4 Member Data Documentation . . . . .	321
16.44.4.1 _opt_vec_idx . . . . .	321
<b>17 File Documentation</b>	<b>323</b>
17.1 base_controller.h File Reference . . . . .	323
17.2 base_edge.h File Reference . . . . .	323
17.3 base_edge.hpp File Reference . . . . .	324
17.4 base_edge_dynamics.h File Reference . . . . .	324
17.5 base_solver.h File Reference . . . . .	324
17.6 base_solver_least_squares.cpp File Reference . . . . .	325
17.7 base_solver_least_squares.h File Reference . . . . .	325
17.8 base_solver_nonlinear_program_dense.cpp File Reference . . . . .	325
17.9 base_solver_nonlinear_program_dense.h File Reference . . . . .	325
17.10bound_constraints.h File Reference . . . . .	326
17.11common_teb_edges.h File Reference . . . . .	326
17.12config.h File Reference . . . . .	327
17.13download.md File Reference . . . . .	328
17.14graph.h File Reference . . . . .	328
17.15groups.dox File Reference . . . . .	329
17.16install.md File Reference . . . . .	329
17.17integrators.h File Reference . . . . .	329
17.18mainpage.dox File Reference . . . . .	329
17.19matlab_class_handle.hpp File Reference . . . . .	329

17.20	matlab_interface.md File Reference . . . . .	329
17.21	measure_cpu_time.h File Reference . . . . .	330
17.21.1	Macro Definition Documentation . . . . .	330
17.21.1.1	START_TIMER . . . . .	330
17.21.1.2	STOP_TIMER . . . . .	330
17.22	misc.h File Reference . . . . .	330
17.22.1	Function Documentation . . . . .	331
17.22.1.1	norm_angle . . . . .	331
17.22.1.2	norm_angle_vec . . . . .	332
17.23	simulator.h File Reference . . . . .	332
17.24	simulator.hpp File Reference . . . . .	333
17.25	solver_levenbergmarquardt_eigen_dense.cpp File Reference . . . . .	333
17.26	solver_levenbergmarquardt_eigen_dense.h File Reference . . . . .	333
17.27	solver_levenbergmarquardt_eigen_sparse.cpp File Reference . . . . .	333
17.28	solver_levenbergmarquardt_eigen_sparse.h File Reference . . . . .	334
17.29	solver_nlopt_package.cpp File Reference . . . . .	334
17.30	solver_nlopt_package.h File Reference . . . . .	334
17.31	solver_overview.md File Reference . . . . .	334
17.32	solver_sqp_dense.h File Reference . . . . .	334
17.32.1	Macro Definition Documentation . . . . .	335
17.32.1.1	__SUPPRESSANYOUTPUT__ . . . . .	335
17.33	solver_sqp_dense_backup.h File Reference . . . . .	335
17.34	system_dynamics.h File Reference . . . . .	335
17.35	teb_controller.h File Reference . . . . .	336
17.36	teb_controller.hpp File Reference . . . . .	336
17.37	teb_plotter.cpp File Reference . . . . .	336
17.38	teb_plotter.h File Reference . . . . .	337
17.39	teb_plotter.hpp File Reference . . . . .	337
17.40	teb_vertices.h File Reference . . . . .	337
17.41	typedefs.h File Reference . . . . .	338
17.41.1	Macro Definition Documentation . . . . .	339
17.41.1.1	INF . . . . .	339
17.41.1.2	INPUT_STREAM . . . . .	339
17.41.1.3	PI . . . . .	339
17.41.1.4	PRINT_DEBUG . . . . .	339
17.41.1.5	PRINT_DEBUG_COND . . . . .	339
17.41.1.6	PRINT_DEBUG_COND_ONCE . . . . .	339
17.41.1.7	PRINT_DEBUG_ONCE . . . . .	340
17.41.1.8	PRINT_ERROR . . . . .	340
17.41.1.9	PRINT_INFO . . . . .	340

17.41.1.10	PRINT_INFO_COND	340
17.41.1.11	PRINT_INFO_COND_ONCE	340
17.41.1.12	PRINT_INFO_ONCE	340
17.42	utilities.h File Reference	340
17.43	workspaces.h File Reference	341
<b>18</b>	<b>Example Documentation</b>	<b>343</b>
18.1	integrator_system1.cpp	343
18.2	integrator_system2.cpp	345
18.3	integrator_system_classic_mpc.cpp	347
18.4	integrator_system_mex.cpp	349
18.5	integrator_system_sfuns.cpp	351
18.6	linear_system_ode.cpp	353
18.7	linear_system_ode_ctrl_comparison.cpp	355
18.8	linear_system_state_space.cpp	357
18.9	mobile_robot_teb.cpp	359
18.10	rocket_system.cpp	368
18.11	van_der_pol_system.cpp	370
	<b>Bibliography</b>	<b>372</b>
	<b>Index</b>	<b>374</b>

# Chapter 1

## Timed-Elastic-Band Package Documentation

### 1.1 Overview

This project contains the implementation of a predictive controller and trajectory planner based on Timed-Elastic-Bands (TEB) (see [5]).

The TEB merges the states, control inputs and time intervals into a joint trajectory representation which enables planning of time-optimal trajectories in the context of model predictive control. Model predictive control integrates the planning of the optimal trajectory with state feedback in the control loop. The TEB approach formulates the fixed horizon optimal control problem for point-to-point transitions as a nonlinear program.

### 1.2 TEB Optimization Problem

#### 1.2.1 Problem Formulation

The TEB joint trajectory representation is defined as follows:

$$\mathcal{B} := \{\mathbf{x}_1, \mathbf{u}_1, \mathbf{x}_2, \mathbf{u}_2, \dots, \mathbf{x}_{n-1}, \mathbf{u}_{n-1}, \mathbf{x}_n, \Delta T\}$$

$\mathbf{x}_i \in \mathbb{R}^p$  and  $\mathbf{u}_i \in \mathbb{R}^q$  are the discretized state and control input vectors respectively.  $\Delta T$  denotes the time difference required to transit from the  $i$ -th state to the consecutive one. Currently, this time difference is obtained from applying finite differences to the system dynamics equation (by replacing continuous derivatives).

This project considers nonlinear programs of the following form to optimize the TEB sequence  $\mathcal{B}$ .

$$V^*(\mathcal{B}) = \min f(\mathcal{B}) \tag{1.1}$$

s.t.:

$$\mathbf{x}_1 = \mathbf{x}_s \tag{1.2}$$

$$\mathbf{x}_n \approx \mathbf{x}_f \tag{1.3}$$

$$\mathbf{h}_k(\mathcal{B}) = \mathbf{0} \quad k \in \mathcal{E} \tag{1.4}$$

$$\mathbf{g}_k(\mathcal{B}) \leq \mathbf{0} \quad k \in \mathcal{I} \tag{1.5}$$

in which  $f(\mathcal{B})$  denotes the objective/cost function. Equation (2) ensures that the nonlinear program constitutes at least an initial value problem. The final constrained equation (3) is not mandatory. But if it is not provided, the final state behavior should be specified using dedicated final state costs as part of  $f(\mathcal{B})$ .  $\mathcal{E}$  and  $\mathcal{I}$  denote the indices sets of equality and inequality constraints respectively.

For many problems the structure of the underlying optimization problem is sparse. To exploit these local relationships between states and control inputs, the optimization problem is formulated as a hyper-graph according to the

formulism in [2]. The user has to define edges that represent "local" cost functions according to [teb::BaseEdge](#) and its subclasses. They can be added to the TEB controller by overriding [teb::TebController::customOptimization-Graph\(\)](#). A lot of common optimization and model predictive control problems are already implemented as Edge (especially for nonlinear dynamic systems in state-space representation. Refer to [teb::SystemDynamics](#) and the example programs for further information.

### 1.2.2 Solution of the TEB Optimization Problem

Solving the nonlinear program (1)-(5) requires the utilization of dedicated solvers that support constraints. The solver classes are constantly expanded as subclasses of [teb::BaseSolver](#).

An approximation to the solution can be computed efficiently using "Least Squares Solvers" and "Soft Constraints". Least Square Optimization Problem require squared objectives. The solver class [teb::BaseSolverLeastSquares](#) transforms equation (1) to  $V^*(\mathcal{B}) = \min f(\mathcal{B})^2$ . Hard constraints (4) and (5) are transformed to soft constraints. Refer to the specific class description. [teb::SolverLevenbergMarquardtEigenDense](#) optimizes this transformed "least-squares" problem using the Levenberg-Marquardt algorithm (Dense means, that no sparse structure is exploited for the solution of the linear system. But only for the calculation of Jacobians due to the hyper-graph representation).

## 1.3 Further Information

For more detailed information about the Timed-Elastic-Band approach and applications refer to [3] [4] [1] [2].

### 1.3.1 Download

Refer to the [download section](#).

### 1.3.2 Installation

Refer to the [installation instructions](#).

#### Author

Christoph Rösmann ([christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de))

#### Remarks

For research only.



## Chapter 2

# Download TEB-Package

### For Users

The project source code is managed by a subversion control system (SVN). Find the svn repository at <https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/>.

To get the newest (experimental and maybe unstable) version checkout the *trunk* folder using:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk teb_package
```

Find versions tagged as stable or *working and tested with windows* in the *tags* folder.

They can be checked out in a similar way (**Warning:** Never commit changes to an existing tagged version, copy it to your own branch instead).

To check out a specific branch of the project, consider the *branches* folder.

### For Developers

#### Work on the trunk

Checkout the trunk:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk teb_package
```

Modify or add code and check if you can compile the code successfully. Check the status of your changes:

```
svn st
```

Check files marked with a question mark, whether they should be added to the repository (use `svn add <path/filename>`).

**Never** add build files and temporary files to the repository!

Afterwards commit your changes to the trunk:

```
svn ci -m "This should be a comprehensive commit message for the log"
```

Next time you continue working on the trunk, don't forget to call `svn up` to get the latest updates.

In case of conflicts, please resolve them by merging the code or contact the author of the previous (conflicting) commit!

#### Create your own branch

*Way 1:*

Create the branch directly on the server:

```
svn cp https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk \
https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch
```

Now checkout your newly created branch. If you have no working copy of the project yet, use:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch teb_mybranch
```

If you already have a working copy (e.g. the trunk folder) call

```
svn switch https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch
```

in the root of your trunk folder.

*Way 2:*

In case you checked out the complete project (including all branches and tags) using:

```
svn co https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/ teb_package
```

Now create a new branch while copying all project files (internally svn is smart enough to prevent double disk space usage):

```
mkdir branches/mybranch
svn add branches/mybranch
svn cp trunk branches/mybranch # Or copy a tag from the tags folder instead of the trunk to switch to the bra
```

### Synchronizing your branch with the trunk

To merge the most recent updates from the trunk into your own branch do the following.

First check what files have been changed compared to your branch:

```
svn st -u
```

Merge the differences in your local working directory:

```
svn merge https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/trunk
```

Often it is recommended to add `--dry-run` to the argument list in order to check for possible conflicts. See the remarks below on manually resolving conflicts.

Finally commit your merged code to your branch.

### Merging a branch into the trunk

First synchronize your branch with the trunk and commit changes like mentioned before.

Now checkout the trunk to a dedicated directory.

Make sure that you do not have any local changes on the trunk (`svn up; svn st`).

Merge the (already synchronized) branch "mybranch" into the trunk using:

```
svn merge --reintegrate https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/branches/mybranch
```

Finally, commit your merged trunk to the server.

[Click here](#) and [here](#) for more information about merging.

### Information about your working directory

Get information on your current revision and if you are working on the trunk or a branch using

```
svn info
```

### Reverting local changes

Undo changes that are not yet committed:

```
svn revert <file>
```

Or undo everything (switch to the main directory first):

```
svn revert --recursive .
```

### **Correcting commit log messages**

If you want to correct a previously made commit message, call:

```
svn propedit svn:log --revprop -r <REV-NR> https://svc.rst.e-technik.tu-dortmund.de/svn/TimedElasticBand/C++/
```

where <REV-NR> denotes the revision number (you can get revision numbers via `svn log`).

### **Resolving conflicts**

If svn was not able to solve conflicts automatically, try to open the files marked as conflicted and search for lines starting with "<<<<<<<" and ">>>>>>>". Edit them manually and afterwards call

```
svn resolved <file>
```

### **Useful symbolic revision names**

For some revisions, special symbolic names can be utilized:

- HEAD: latest (youngest) revision on the server
- BASE: the revision your checkout was last updated against
- COMMITTED: last revision a file or directory was changed
- PREV: the last revision the file or directory was changed immediatley before COMMITTED



## Chapter 3

# Install TEB-Package

### Ubuntu

#### Prerequisites:

- *CMake*: Required to create the platform dependent build files (`sudo apt-get install cmake`).
- *Gnuplot* (optional, but recommended): Required for plotting and visualization (`sudo apt-get install gnuplot gnuplot-x11`).
- *Doxygen* (optional): Necessary, if the documentation should be compiled (`sudo apt-get install doxygen`).
- *Tex Live* (optional): Only necessary to create a pdf documentation from the latex source generated by doxygen.
- *Matlab* (optional): Necessary only if the user wants to create mex-files and s-functions for the usage in Matlab and Simulink.
- *kDevelop* (optional): IDE for developers. It supports the import CMakeLists.txt directly (`sudo apt-get install kdevelop`).
- *SuiteSparse* (optional): Enables the usage of a wider range of sparse direct linear solvers (`sudo apt-get install libsuitesparse-dev`).

#### Usage:

TODO

#### Doxygen:

After the project is generated using CMake, cd into the `./build` folder and call `make doc` from the terminal. Now a HTML and a LATEX folder should be created inside the doc folder. To create a PDF from the LATEX source directly, cd to `./doc/LATEX` and type in your terminal: `make pdf` (Tex Live including pdflatex required!).

### Mac OSX

#### Prerequisites:

- *XCode*: Download the newest x-code version from the app store or from the OS X developer page.
- *CMake*: Required to create the platform dependent build files (<http://www.cmake.org>).
- *Gnuplot* (optional, but recommended): Required for plotting and visualization (<http://gnuplot.info>)
- *Doxygen* (optional): Necessary, if the documentation should be compiled (<http://www.stack.nl/~dimitri/doxygen/download.html>).

- *MacTex* (optional): Only necessary to create a pdf documentation from the latex source generated by doxygen.
- *Matlab* (optional): Necessary only if the user wants to create mex-files and s-functions for the usage in Matlab and Simulink.

**Remarks:** For me (Christoph), x11 (xquartz) was not working if gnuplot was called from default terminal. It worked only in xterm. Since xcode starts the default terminal, I decided to use aquaterm in the TEBPlotter class:

```
install xcode with command line tools
install homebrew
install aquaterm
terminal: brew install gnuplot --with-aquaterm
test installation:
terminal: gnuplot
plot x
```

### Usage:

Open CMake and select the CMakeLists.txt inside the project folder. Select x-code as the underlying compiler, configure and generate the project into a "build" folder. You can find the x-code project files now inside the newly created build folder.

### Doxygen:

After the project is generated using CMake, cd into the ./doc folder and call `doxygen doxygen.config` from the terminal. Now a HTML and a LATEX folder should be created inside the doc folder. To create a PDF from the LATEX source directly, cd to ./doc/LATEX and type in your terminal: `make pdf` (MacTex including pdflatex required!).

## Windows

### Prerequisites:

- *Visual Studio 2013+:* Download Visual Studio (At least Version 12 / 2013)
- *CMake:* Required to create the platform dependent build files (<http://www.cmake.org>).
- *Gnuplot* (optional, but recommended): Required for plotting and visualization (<http://gnuplot.info>)
- *Doxygen* (optional): Necessary, if the documentation should be compiled (<http://www.stack.nl/~dimitri/doxygen/download.html>).
- *MikTex* (optional): Only necessary to create a pdf documentation from the latex source generated by doxygen.
- *Matlab* (optional): Necessary only if the user wants to create mex-files and s-functions for the usage in Matlab and Simulink.

**Remarks:** Gnuplot has to be included in Windows Path variable:

- Windows 8: System Control -> Extended System Settings -> Advanced -> Environment variables. Modify the "System Variable" PATH and append the Gnuplot binary directory. E.g.: PATH;C:\Program Files\gnuplot\bin\

### Usage:

Open CMake and select the C++ project folder. Select Visual Studio 12 (2013) 64 Bit as the underlying compiler, configure and generate the project into a specified "build" folder. **Caution:** We observed problems using 32 Bit, therefore we recommend to select 64 Bit Visual Studio!! You can find the Visual-Studio project files now inside the newly created build folder.

## Chapter 4

# Solver Overview

The following list contains descriptions and small guidances about the solver backends implemented or interfaced within this package.

### Least-Squares Solver

Least-Square solvers minimize only quadratic functions without constraints  $V^*(\mathcal{B}) = \min f^2(\mathcal{B})$ .

Advantages of the least-squares solver are the amazing performance, the robustness (Levenberg-Marquardt) and even after a few iterations, a suitable solution is found often. A drawback is the lack of hard constraints. This implementation transforms equality and inequality constraints automatically into soft constraints (refer to the class description).

Due to the prerequisite that only squared function can be minimized, we need to keep the following important definitions/conventions in mind:

- The objective functions are always defined without squaring the values:  $f(\mathcal{B})$  than  $f^2(\mathcal{B})$ .
- The solver will square the values internally (this definition is much more efficient than specifying the squared values).
- Constraints are squared internally as well using the soft-constraint approximation.
- To make the MPC optimization problem simultaneously compatible to Non-Linear Program Solvers (see below), the designer of an objective function (given as an edge type for the hyper-graph, see the mobile robot example) can specify some information about the function type (see Enumeration [teb::FUNCT\\_TYPE](#)). This can be helpful for the hessian calculation as well, since a linear function has a zero hessian. The function types are defined as follow:
  - *Nonlinear*: The function is nonlinear (it will be squared by LeastSquares solver, but not by other solvers)
  - *Nonlinear\_Squared*: The function is nonlinear (it will be squared by all solvers)
  - *Nonlinear\_Once\_Diff*: The hessian of this function is always zero (and it will be squared only by Least-Square solvers)
  - *Quadratic*: The function will be squared only by LeastSquares solvers (-> Power of 4 !!!)
  - *Linear*: The function will be squared only by LeastSquares solvers (zero hessian)
  - *LinearSquared*: The function will be squared by all solvers (zero hessian)

### Levenberg-Marquardt Dense ([teb::SolverLevenbergMarquardtEigenDense](#))

This solver constitutes a robust least-squares optimization algorithm. The Levenberg-Marquardt algorithm is similar to the one used in [2]. The underlying linear system is solved using the Eigen framework.

### Levenberg-Marquardt Sparse ([teb::SolverLevenbergMarquardtEigenSparse](#))

This solver implements the sparse version of the Levenberg-Marquardt algorithm mentioned above. Jacobians and Hessians are internally treated as sparse matrices, that are efficiently constructed using the hyper-graph formulation of the underlying optimization problem. The resulting sparse linear system is solved using the Eigen framework (sparse modules). If CMake is able to find SuiteSparse/Cholmod libraries (see [Install TEB-Package](#)), the cholmod solver is utilized for solving the sparse linear system as long as `!defined(FORCE_EIGEN_SOLVER)`.

## Nonlinear Program Solver

### Sequential-Quadratic-Programming (SQP) Dense with qpOASES ([teb::SolverSQPDense](#))

The solver implementation is still in progress, documentation will be added in the future.

### NLOPT Solver Interface ([teb::SolverNloptPackage](#))

The solver implementation is still in progress, documentation will be added in the future.



## Chapter 5

# Matlab Interface

The following instructions give you an introduction on how to interface this package with Matlab and Matlab Simulink. The procedure is still experimental, please inform us about bugs and/or possible solutions.

### Way 1: Compile MEX and S-Function wrapper using CMAKE

Advantages:

- Simple solution
- Allows the library *teb\_package.lib* to be compiled either in Debug or Release mode

Disadvantages:

- In order to use the wrappers within *Matlab's/Simulink's RealTime Workshop*, the code must be compilable with Matlab. In this case refer to *Way 2* instead.

You can directly include MEX or S-Function wrapper functions (defined in separate \*.cpp files) into the CMAKE build progress. See the *CMakeLists.txt* for examples.

If the projects are stored in the example folder, add them to:

```
SET(MEX_EXAMPLE_NAMES # matlab mex example files
    matlab_interface/integrator_system_mex
    # Add files without .cpp extension here, attach folder name as a prefix
)

SET(SFUN_EXAMPLE_NAMES # matlab simulink mex example files
    matlab_interface/integrator_system_sfunk
    # Add files without .cpp extension here, attach folder name as a prefix
)
```

If you use a separate project, add them using `ADD_MEX_FILE` macro (see *cmake/macros* directory) and link against the *teb\_library.lib*. You need to include Matlabs source and libs (see *FindMatlab.cmake* in *cmake/modules* and its application in *CMakeLists.txt*).

For information about the MEX wrapper source and the S-Function wrapper source themselves, please refer to the example section.

The TEB library *teb\_package.lib* and the compiled MEX files are stored somewhere inside the build folder, depending on the platform used. Please check *build/bin/lib* and *\*/build/lib/\** first.

### Way 2: Compile MEX and S-Function wrapper using Matlab itself

Advantages:

- Avoids copying the MEX file from the build path to the working directory, since Matlab creates it directly in the current directory.
- Allows compiling with *Matlab's/Simulink's Realtime Workshop*

Disadvantages:

- Allows the library *teb\_package.lib* to be compiled only in Release mode.
- A second independent build process needs to be specified that requires information about libraries, include directories and additional source files.

Choosing this way requires the user to define settings for compiling a MEX wrapper in Matlab. First make sure to have at least *Visual Studio 2013* installed. Check if you set up your compiler in Matlab correctly:

```
mex -setup
```

Now switch to your working directory containing the MEX wrapper source file (For the sake of simplicity, we assume that additional header files are stored in the same folder, otherwise add the corresponding path to build process).

Compile your source code using `mex '-IC:\path-to-include' '-LC:\path-to-library' '-ILibName' 'source-file-name.cpp'`. If you want to use variables, use the following syntax and exchange the strings with variables:

```
mex(['-I' 'C:\teb_package\include'],...           // add project include dir
    ['-I' 'C:\teb_package\extern\eigen3'],...      // add path to Eigen-lib headers
    ['-L' 'C:\teb_package\build\lib\Release'],...  // path to the package library teb_package.lib
    ['-l' 'teb_package'],...                      // the filename of the lib (without extension)
    'integrator_system_mex.cpp');                 // Source file for the MEX or S-Function wrapper.
```

If everything was successful you should now see a compiled MEX-file in your working directory. Please refer to the example section for more details on how to use the MEX/S-Function interface.

### Use your code in Simulink

Create your model and add the S-Function block where you can specify the name of your compiled MEX S-Function. You can use your model in *Simulink Normal Mode* without Compiling

### Compile your code in Simulink

If you want to compile your code in Simulink, go to *Simulation/Model Configuration Parameters/Code Generation* and switch the Language to C++. Select the *Interface* subsection and deselect *MAT-file logging*. Finally, you need to provide all build informations that was required by the mex compiler. Goto the *Custom Code* subsection. In the fragment *Include list of additional* add all paths and files that are required (find here the integrator example):

- Include directories:  
`C:\teb_package\include`  
`C:\teb_package\extern\eigen3`
- Libraries:  
`C:\teb_package\build\lib\Release\teb_package.lib`

Instead of linking the *teb\_package.lib*, you can add the C++ source files (e.g. to avoid *error LNK2038* (see below)) of the package directly:

- Source files: `C:\teb_package\src\base\base_controller.cpp`  
`C:\teb_package\src\base\blue_solver_least_squares.cpp`  
`C:\teb_package\src\base\workspaces.cpp`  
`C:\teb_package\src\solver\solver_levenbergmarquardt_eigen_dense.cpp`  
`C:\teb_package\src\solver\solver_levenbergmarquardt_eigen_sparse.cpp`

Now you can compile your code directly within Simulink. If you have any troubles, please check the *Troubleshooting* section first.

### Compile your code for Simulink xPC Target

Go to *Simulation/Model Configuration Parameters/Code Generation* and change the target to *Simulink Real-Time*. Afterwards follow the steps mentioned in *Compile your code in Simulink* above. But use the source files method instead of linking the precompiled library *teb\_package.lib* before.

Simulink xPC Target does not support IO. Therefore you have to define *RTW* before compiling the *teb\_package* / model. Define it using the CXX Flag *\*-DRTW\** or uncomment the line *'#define RTW'* in the file [typedefs.h](#).

Now you can compile you should be able to compile your model for xPC target.

### Compile your code for Simulink Real-Time Windows Target

Compilation for RTWIN is currently only supported in *Normal Mode*. Compiling in *External Mode* invokes a dedicated Real-Time Kernel that does not support dynamic memory allocation and exceptions. Those are both utilized in the C++ standard library (STL) which is widely used within this project.

### Troubleshooting

- *error LNK2038: mismatch detected for '\_ITERATOR\_DEBUG\_LEVEL': value '2' doesn't match value '0'*  
Try to compile the TEB package in Release mode: Check first lines in *CMakeLists.txt*!
- *error LNK2038: Konflikt ermittelt für "RuntimeLibrary": Der Wert "MD\_DynamicRelease" stimmt nicht mit dem Wert "MT\_StaticRelease" xxx\_sfun.obj überein.*

Simulink compiles determines that some parts are compiled as static and some other parts as dynamic, therefore the linker cannot link them. For me it remains unclear, since the *teb\_package.lib* is compiled statically and the mex compiler is Visual Studio in both cases. You can bypass this error by modifying the compiler settings in *Simulation/Model Configuration Parameters/Code Generation* by changing the *Build configuration* to *Specify* and append the *\*/MD\** flag to the C++ Compiler flags. If you have still problems, try to exchange *\*/MD\** with *\*/MT\**. See the image below: An alternative way to solve the problem is to directly compile the whole *teb\_package* by providing the *\*.cpp* source files instead of the library *teb\_package.lib*.



## Chapter 6

## Test List

### Class `teb::SolverNloptPackage`

This class needs more testing (different solvers etc.)

Member `teb::TebPlotter::plotTwoCol` (`const Eigen::Ref< const Eigen::VectorXd > &col1_time`, `const Eigen::Ref< const Eigen::MatrixXd > &col1_data`, `const Eigen::Ref< const Eigen::VectorXd > &col2_time`, `const Eigen::Ref< const Eigen::MatrixXd > &col2_data`, `PlotOptions *options=nullptr`)

What happens if the right column contains more rows than the left one? (Alignment)



## Chapter 7

# Todo List

### Class `teb::BaseController`

Implement complete API (abstract functions, etc.) to allow a generic controller usage for `teb::Simulator` and `teb::TebPlotter`

### Member `teb::BaseEdge< D, FuncType >::allocateMemory (bool skip_hessian=false)`

Implement automatic memory allocation instead calling this function manually.

Parameters

<code>skip_hessian</code>	If no hessian calculation is required by the solver skip memory allocation.
---------------------------	---

### Member `teb::BaseEdge< D, FuncType >::calculateVertexDimensions ()`

Maybe implement another strategy to store the dimensions.

### Member `teb::BaseEdge< D, FuncType, Vertices >::allocateMemory (bool skip_hessian=false)`

Implement automatic memory allocation instead calling this function manually.

Parameters

<code>skip_hessian</code>	If no hessian calculation is required by the solver skip memory allocation.
---------------------------	---

### Member `teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ()`

Maybe implement static/constexpr version of collecting all dimensions since they are known at compile-time (maybe <http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array-in>)

### Member `teb::BaseSolver::solve (HyperGraph *optimizable_graph)`

Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

### Member `teb::BaseSolverLeastSquares::getChi2 () const`

Maybe switch to the formulation  $f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \Omega \mathbf{f}(\mathcal{B})$  similar to g2o and Teb-Matlab.

### Class `teb::BaseSolverNonlinearProgramDense`

The documentation of this class, after this class passed a couple of more tests

### Class `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >`

Maybe use a special representation or zero Hessians to avoid processing a lot of zeros (same for other edges)

### Class `teb::EdgeQuadraticForm< p, q >`

Currently the Levenberg-Marquardt solver takes the cost function squared by itself, therefore we need to implement the sqrt here: For other solvers this cost function will be problematic without taking the square.

### Class `teb::HessianWorkspace`

`TebVertex::dimension()` is used at the moment. Actually only a number of `TebVertex::dimensionFree()` non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

**Class `teb::JacobianWorkspace`**

`TebVertex::dimension()` is used at the moment. Acutally only a number of `TebVertex::dimensionFree()` non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

**Class `teb::NumericalIntegrator< p, q >`**

Implement and test Runge Kutta with derivatives <http://www.emis.de/journals/EJDE/conf-proc/02/g1/goek.pdf>

**Class `teb::PlotOptions`**

Extend functionality

**Class `teb::Simulator< p, q >`**

Add disturbance models.

Add reference trajectory or multiple reference steps for closed-loop control.

**Member `teb::Simulator< p, q >::setSampleTime (double sample_time)`**

Allow different sample\_times for controller and plant simulation (the TEB sample time  $\Delta T$  might be higher ...)

**Class `teb::SolverLevenbergMarquardtEigenSparse`**

Consider only the upper diagonal part of the hessian?

**Member `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian ()`**

Check if implemented ordering of column and row iterations fits Eigen's column major representation best.

**Class `teb::SolverSQPDense`**

The documentation of this class, after the SQP solver passed a couple of remaining (robustness) tests

**Class `teb::SystemDynamics< p, q >`**

Methods/Intefrace for jacobian and hessian calculation

**Class `teb::TebController< p, q >`**

Using the `TebController` inside a standard container (e.g. vector) does not work as supposed to do.

Create copy and move constructors.

**Member `teb::TebController< p, q >::activateControlBounds (unsigned int control_idx, double u_min, double u_max, bool activate=true)`**

Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

**Member `teb::TebController< p, q >::activateStateBounds (unsigned int state_idx, double x_min, double x_max, bool activate=true)`**

Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

**Member `teb::TebController< p, q >::activateStateBounds (const Eigen::Ref< const StateVector > &x_min, const Eigen::Ref< const StateVector > &x_max, bool activate=true)`**

Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

**Member `teb::TebController< p, q >::buildOptimizationGraph ()`**

Here we need to fix the last control input since it undefined (no control is needed to go to state  $n+1$ ). Maybe we can change the code somewhere else, that the contorl sequence is always `getN()-1`.

**Member `teb::TebController< p, q >::firstControlSaturated () const`**

Support multiple control input bounds (now its scalar for all elements of `u`).



---

#### Returns

Saturated control input  $u_k$  [ $q \times 1$ ].

#### Member `teb::TebController< p, q >::getActiveVertices ()`

Add case for different state and control input sequence lengths (e.g. for move blocking)

#### Member `teb::TebController< p, q >::initTrajectory (const Eigen::Ref< const StateVector > &x0, const Eigen::Ref< const StateVector > &xf, unsigned int n)`

More efficient implementation that keeps previously allocated memory

#### Member `teb::TebController< p, q >::optimizeTEB ()`

The current implementation is really inefficient, since in each outer-loop-iteration all hyper-graph edges are deleted and afterwards reinstantiated. Create a dedicated update function in the future to hot-start from a previous hyper-graph.

#### Member `teb::TebController< p, q >::resampleTrajectory (unsigned int n_new)`

More efficient strategy without copying the complete sequences.

: later this should be  $n_{\text{new}}-1$

#### Member `teb::TebController< p, q >::returnControllInputSequence () const`

Here we assume, that the last control input is invalid - We need to change this, if we have different sizes for state and control input sequences



## Chapter 8

# Bug List

**Member** [teb::Config::Optim::Solver::Lsq::soft\\_constr\\_epsilon](#)

Do not use at the moment, because it is not implemented correctly

**Class** [teb::EdgeSystemDynamics< p, q, central >](#)

Central differences do not seem to work as expected at the moment.

**Member** [teb::Simulator< p, q >::simClosedLoop](#) (const double \*const x0, const double \*const xf, double sim\_time, bool manual\_stepping=false, bool plot\_result=true, bool plot\_steps=true, BaseController \*ctrl=nullptr)

If we set a title or legend in the plot options for stepping (!), than the plot won't be rendered correctly.

**Class** [teb::TebPlotter](#)

EPS and PDF export are not working as expected. No text is displayed and after importing to Inksape, the plot is blank (noticed on Ubuntu 14.04).



## Chapter 9

# Module Index

### 9.1 Modules

Here is a list of all modules:

Controller . . . . .	??
Solver . . . . .	??
Simulation . . . . .	??
Visualization . . . . .	??



## Chapter 10

# Namespace Index

### 10.1 Namespace List

Here is a list of all namespaces with brief descriptions:

[teb](#)

General project namespace for all library functions and objects . . . . . ??





## Chapter 11

# Hierarchical Index

### 11.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

teb::BaseController	??
teb::TebController< p, q >	??
teb::BaseSolver	??
teb::BaseSolverLeastSquares	??
teb::SolverLevenbergMarquardtEigenDense	??
teb::SolverLevenbergMarquardtEigenSparse	??
teb::BaseSolverNonlinearProgramDense	??
teb::SolverSQPDense	??
teb::SolverSQPDense	??
teb::SolverNloptPackage	??
teb::Config	??
teb::CustomBoundData	??
teb::EdgeType	??
teb::LINEAR, Vertex >	??
teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >	??
teb::EdgeMinimizeTime	??
teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >	??
teb::EdgeQuadraticForm< p, q >	??
teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >	??
teb::EdgeSystemDynamics< p, q, central >	??
teb::BaseEdge< D, FuncType, Vertices >	??
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >	??
teb::BaseEdge< D, FuncType >	??
teb::Config::Optim::Solver::NonlinearProgram::Hessian	??
teb::HessianWorkspace	??
teb::HyperGraph	??
teb::JacobianWorkspace	??
teb::Config::Optim::Solver::NonlinearProgram::LineSearch	??
teb::Config::Optim::Solver::Lsq	??
teb::Config::Optim::Solver::Nlopt	??
teb::Config::Optim::Solver::NonlinearProgram	??
teb::NumericalIntegrator< p, q >	??
teb::ExplicitEuler< p, q >	??
teb::RungeKutta5thOrder< p, q >	??
teb::RungeKuttaClassic< p, q >	??

teb::Config::Optim . . . . .	??
teb::PlotOptions . . . . .	??
teb::SimResults . . . . .	??
teb::Simulator< p, q > . . . . .	??
teb::Config::Optim::Solver . . . . .	??
teb::SystemDynamics< p, q > . . . . .	??
teb::Config::Teb . . . . .	??
teb::TebPlotter . . . . .	??
teb::SimResults::TimeSeries . . . . .	??
teb::Config::Utilities . . . . .	??
teb::VertexType . . . . .	??
teb::ControlVertex< q > . . . . .	??
teb::StateVertex< p > . . . . .	??
teb::TimeDiff . . . . .	??

## Chapter 12

# Class Index

### 12.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">teb::BaseController</a>	Base controller class (General Controller API) . . . . .	??
<a href="#">teb::BaseEdge&lt; D, FuncType, Vertices &gt;</a>	Templated base edge class that stores an arbitrary number of values . . . . .	??
<a href="#">teb::BaseEdge&lt; D, FuncType &gt;</a>	Templated base edge class that stores an arbitrary number of values (Partial template specialization that allows changing vertices dimensions at runtime) . . . . .	??
<a href="#">teb::BaseSolver</a>	Base class for solver implementations . . . . .	??
<a href="#">teb::BaseSolverLeastSquares</a>	Extended base solver class for least squares optimizations . . . . .	??
<a href="#">teb::BaseSolverNonlinearProgramDense</a>	Extended base solver class for nonlinear programs (dense version) . . . . .	??
<a href="#">teb::BoundConstraint&lt; Bound_type, Vertex, Bound_vars, Index &gt;</a>	This class captures general bound constraints on optimization variables / vertices . . . . .	??
<a href="#">teb::Config</a>	Configurations for Controller and Solver classes . . . . .	??
<a href="#">teb::ControlVertex&lt; q &gt;</a>	Vertex that contains the ControlVector . . . . .	??
<a href="#">teb::CustomBoundData</a>	Datastruct for custom bound data that can be queried from the <a href="#">BoundConstraint</a> class . . . . .	??
<a href="#">teb::EdgeMinimizeTime</a>	Objective edge: minimize $\Delta T$ . . . . .	??
<a href="#">teb::EdgeQuadraticForm&lt; p, q &gt;</a>	Objective edge: quadratic form for states and control input . . . . .	??
<a href="#">teb::EdgeSystemDynamics&lt; p, q, central &gt;</a>	Equality constraint edge for satisfying continious system dynamics specified by an <a href="#">SystemDynamics</a> object . . . . .	??
<a href="#">teb::EdgeType</a>	Generic interface class for edges . . . . .	??
<a href="#">teb::ExplicitEuler&lt; p, q &gt;</a>	Simple explicit euler method for integration . . . . .	??
<a href="#">teb::Config::Optim::Solver::NonlinearProgram::Hessian</a>	Configurations for the hessian of the lagrangian calculation . . . . .	??
<a href="#">teb::HessianWorkspace</a>	Memory management and accessor functions for the Block-Hessian (for each edge) . . . . .	??
<a href="#">teb::HyperGraph</a>	Graph object that stores pointers to active vertices and edges . . . . .	??

<a href="#">teb::JacobianWorkspace</a>	Memory management and accessor functions for the Block-Jacobians (for each edge) . . . . .	??
<a href="#">teb::Config::Optim::Solver::NonlinearProgram::LineSearch</a>	Settings for the line-search algorithm (force merit function descent) . . . . .	??
<a href="#">teb::Config::Optim::Solver::Lsq</a>	Configurations especially for least-squares-solvers . . . . .	??
<a href="#">teb::Config::Optim::Solver::Nlopt</a>	Configurations especially for the nlopt solver wrapper . . . . .	??
<a href="#">teb::Config::Optim::Solver::NonlinearProgram</a>	Settings for constrained optimization solver (nonlinear program solver) . . . . .	??
<a href="#">teb::NumericalIntegrator&lt; p, q &gt;</a>	Interface for numerical integration methods used in simulation . . . . .	??
<a href="#">teb::Config::Optim</a>	Configurations related to the trajectory optimization . . . . .	??
<a href="#">teb::PlotOptions</a>	Customize plots and figures . . . . .	??
<a href="#">teb::RungeKutta5thOrder&lt; p, q &gt;</a>	Runge Kutta method (fifth order, slightly modified) . . . . .	??
<a href="#">teb::RungeKuttaClassic&lt; p, q &gt;</a>	Classic Runge Kutta method (fourth order) . . . . .	??
<a href="#">teb::SimResults</a>	Simulation results are stored and combined in this container class . . . . .	??
<a href="#">teb::Simulator&lt; p, q &gt;</a>	<a href="#">Simulator</a> for dynamic systems controlled by a <a href="#">TebController</a> . . . . .	??
<a href="#">teb::Config::Optim::Solver</a>	Configurations related to solvers . . . . .	??
<a href="#">teb::SolverLevenbergMarquardtEigenDense</a>	Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version) . . . . .	??
<a href="#">teb::SolverLevenbergMarquardtEigenSparse</a>	Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version) . . . . .	??
<a href="#">teb::SolverNloptPackage</a>	This class wraps the optimization problem to allow solving by NLOPT . . . . .	??
<a href="#">teb::SolverSQPDense</a>	Dense sequential quadratic programming (SQP) solver for nonlinear programs . . . . .	??
<a href="#">teb::StateVertex&lt; p &gt;</a>	Vertex that contains the StateVector . . . . .	??
<a href="#">teb::SystemDynamics&lt; p, q &gt;</a>	Helper class for modeling nonlinear dynamic systems . . . . .	??
<a href="#">teb::Config::Teb</a>	Configurations related to the TEB algorithm . . . . .	??
<a href="#">teb::TebController&lt; p, q &gt;</a>	Main Timed-Elastic-Band controller class . . . . .	??
<a href="#">teb::TebPlotter</a>	Provides useful visualization and plotting functions . . . . .	??
<a href="#">teb::TimeDiff</a>	Vertex that contains the time information of the TEB: $\Delta T$ . . . . .	??
<a href="#">teb::SimResults::TimeSeries</a>	Store measurements of dynamic systems (states and control inputs) w.r.t . . . . .	??
<a href="#">teb::Config::Utilities</a>	Configurations for helper miscellaneous and utilities . . . . .	??
<a href="#">teb::VertexType</a>	Generic interface class for vertices . . . . .	??

## Chapter 13

# File Index

### 13.1 File List

Here is a list of all files with brief descriptions:

<a href="#">base_controller.h</a>	??
<a href="#">base_edge.h</a>	??
<a href="#">base_edge.hpp</a>	??
<a href="#">base_edge_dynamics.h</a>	??
<a href="#">base_solver.h</a>	??
<a href="#">base_solver_least_squares.cpp</a>	??
<a href="#">base_solver_least_squares.h</a>	??
<a href="#">base_solver_nonlinear_program_dense.cpp</a>	??
<a href="#">base_solver_nonlinear_program_dense.h</a>	??
<a href="#">bound_constraints.h</a>	??
<a href="#">common_teb_edges.h</a>	??
<a href="#">config.h</a>	??
<a href="#">graph.h</a>	??
<a href="#">integrators.h</a>	??
<a href="#">matlab_class_handle.hpp</a>	??
<a href="#">measure_cpu_time.h</a>	??
<a href="#">misc.h</a>	??
<a href="#">simulator.h</a>	??
<a href="#">simulator.hpp</a>	??
<a href="#">solver_levenbergmarquardt_eigen_dense.cpp</a>	??
<a href="#">solver_levenbergmarquardt_eigen_dense.h</a>	??
<a href="#">solver_levenbergmarquardt_eigen_sparse.cpp</a>	??
<a href="#">solver_levenbergmarquardt_eigen_sparse.h</a>	??
<a href="#">solver_nlopt_package.cpp</a>	??
<a href="#">solver_nlopt_package.h</a>	??
<a href="#">solver_sqp_dense.h</a>	??
<a href="#">solver_sqp_dense_backup.h</a>	??
<a href="#">system_dynamics.h</a>	??
<a href="#">teb_controller.h</a>	??
<a href="#">teb_controller.hpp</a>	??
<a href="#">teb_plotter.cpp</a>	??
<a href="#">teb_plotter.h</a>	??
<a href="#">teb_plotter.hpp</a>	??
<a href="#">teb_vertices.h</a>	??
<a href="#">typedefs.h</a>	??
<a href="#">utilities.h</a>	??
<a href="#">workspaces.h</a>	??

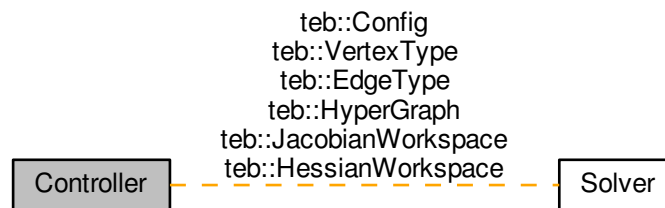


## Chapter 14

# Module Documentation

### 14.1 Controller

Collaboration diagram for Controller:



### Classes

- class `teb::BaseController`  
*Base controller class (General Controller API)*
- class `teb::BaseEdge< D, FuncType, Vertices >`  
*Templated base edge class that stores an arbitrary number of values.*
- class `teb::BaseEdge< D, FuncType >`  
*Templated base edge class that stores an arbitrary number of values (Partial template specialization that allows changing vertices dimensions at runtime).*
- class `teb::EdgeSystemDynamics< p, q, central >`  
*Equality constraint edge for satisfying continuous system dynamics specified by an [SystemDynamics](#) object.*
- class `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >`  
*This class captures general bound constraints on optimization variables / vertices.*
- class `teb::EdgeMinimizeTime`  
*Objective edge: minimize  $\Delta T$ .*
- class `teb::EdgeQuadraticForm< p, q >`  
*Objective edge: quadratic form for states and control input.*
- class `teb::Config`  
*Configurations for Controller and Solver classes.*

- class [teb::VertexType](#)  
*Generic interface class for vertices.*
- class [teb::EdgeType](#)  
*Generic interface class for edges.*
- class [teb::HyperGraph](#)  
*Graph object that stores pointers to active vertices and edges.*
- class [teb::SystemDynamics< p, q >](#)  
*Helper class for modeling nonlinear dynamic systems.*
- class [teb::TebController< p, q >](#)  
*Main Timed-Elastic-Band controller class.*
- class [teb::StateVertex< p >](#)  
*Vertex that contains the StateVector.*
- class [teb::ControlVertex< q >](#)  
*Vertex that contains the ControlVector.*
- class [teb::TimeDiff](#)  
*Vertex that contains the time information of the TEB:  $\Delta T$ .*
- class [teb::JacobianWorkspace](#)  
*Memory management and accessor functions for the Block-Jacobians (for each edge).*
- class [teb::HessianWorkspace](#)  
*Memory management and accessor functions for the Block-Hessian (for each edge).*

## Typedefs

- using [teb::EdgePositiveTime](#) = BoundConstraint< BOUND\_TYPE::LOWER, TimeDiff, BOUND\_VARS::SINGLE, 0 >  
*Bound constraint edge for the time difference:  $\Delta T \geq \varepsilon$ .*
- template<int q>  
using [teb::EdgeControlBounds](#) = BoundConstraint< BOUND\_TYPE::LOWERUPPER, ControlVertex< q >, BOUND\_VARS::ALL >  
*Bound constraint edge for control inputs:  $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$ .*
- template<int p>  
using [teb::EdgeStateBounds](#) = BoundConstraint< BOUND\_TYPE::LOWERUPPER, StateVertex< p >, BOUND\_VARS::ALL >  
*Bound constraint edge for states:  $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$ .*

### 14.1.1 Detailed Description

### 14.1.2 Typedef Documentation

- 14.1.2.1 `template<int q> using teb::EdgeControlBounds = typedef BoundConstraint<BOUND_TYPE::LOWERUPPER, ControlVertex<q>, BOUND_VARS::ALL>`

The following two inequality constraints are implemented:

$$\mathbf{u}_k \geq \mathbf{u}_{min} \quad (14.1)$$

$$\mathbf{u}_k \leq \mathbf{u}_{max} \quad (14.2)$$

$\mathbf{u}_{min}$  and  $\mathbf{u}_{max}$  are the lower and upper bounds on the control input  $\mathbf{u}_k$ .

Vertices required (connect the edge with the following vertices):

1. [ControlVertex](#)  $\mathbf{u}_k$



## Template Parameters

$p$	Number of state variables
$q$	Number of input variables

Definition at line 111 of file common\_teb\_edges.h.

14.1.2.2 `using teb::EdgePositiveTime = typedef BoundConstraint<BOUND_TYPE::LOWER, TimeDiff, BOUND_VARS::SINGLE, 0>`

The following inequality constraint is implemented:

$$\Delta T \geq \varepsilon$$

$\varepsilon$  denotes an offset.

Set the offset  $\varepsilon$  using `setBounds()`.

Vertices required (connect the edge with the following vertices):

1. [TimeDiff](#)  $\Delta T$

Definition at line 89 of file common\_teb\_edges.h.

14.1.2.3 `template<int p> using teb::EdgeStateBounds = typedef BoundConstraint<BOUND_TYPE::LOWERUPPER, StateVertex<p>, BOUND_VARS::ALL>`

The following two inequality constraints are implemented:

$$\mathbf{x}_k \geq \mathbf{x}_{min} \quad (14.3)$$

$$\mathbf{x}_k \leq \mathbf{x}_{max} \quad (14.4)$$

$\mathbf{x}_{min}$  and  $\mathbf{x}_{max}$  are the lower and upper bounds on the state vector  $\mathbf{x}_k$ .

Vertices required (connect the edge with the following vertices):

1. [StateVertex](#)  $\mathbf{x}_k$

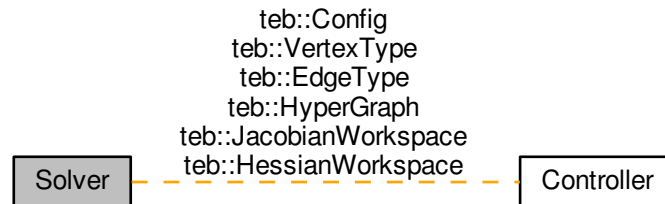
## Template Parameters

$p$	Number of state variables
$q$	Number of input variables

Definition at line 133 of file common\_teb\_edges.h.

## 14.2 Solver

Collaboration diagram for Solver:



### Classes

- class [teb::BaseSolver](#)  
*Base class for solver implementations.*
- class [teb::BaseSolverLeastSquares](#)  
*Extended base solver class for least squares optimizations.*
- class [teb::Config](#)  
*Configurations for Controller and Solver classes.*
- class [teb::VertexType](#)  
*Generic interface class for vertices.*
- class [teb::EdgeType](#)  
*Generic interface class for edges.*
- class [teb::HyperGraph](#)  
*Graph object that stores pointers to active vertices and edges.*
- class [teb::JacobianWorkspace](#)  
*Memory management and accessor functions for the Block-Jacobians (for each edge).*
- class [teb::HessianWorkspace](#)  
*Memory management and accessor functions for the Block-Hessian (for each edge).*
- class [teb::BaseSolverNonlinearProgramDense](#)  
*Extended base solver class for nonlinear programs (dense version).*
- class [teb::SolverLevenbergMarquardtEigenDense](#)  
*Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).*
- class [teb::SolverLevenbergMarquardtEigenSparse](#)  
*Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).*
- class [teb::SolverNloptPackage](#)  
*This class wraps the optimization problem to allow solving by NLOPT.*
- class [teb::SolverSQPDense](#)  
*Dense sequential quadratic programming (SQP) solver for nonlinear programs.*

### 14.2.1 Detailed Description

## 14.3 Simulation

### Classes

- class [teb::NumericalIntegrator< p, q >](#)  
*Interface for numerical integration methods used in simulation.*
- class [teb::ExplicitEuler< p, q >](#)  
*Simple explicit euler method for integration.*
- class [teb::RungeKuttaClassic< p, q >](#)  
*Classic Runge Kutta method (fourth order)*
- class [teb::RungeKutta5thOrder< p, q >](#)  
*Runge Kutta method (fifth order, slightly modified)*
- class [teb::SimResults](#)  
*Simulation results are stored and combined in this container class.*
- class [teb::Simulator< p, q >](#)  
*[Simulator](#) for dynamic systems controlled by a [TebController](#).*

### 14.3.1 Detailed Description

## 14.4 Visualization

### Classes

- struct [teb::PlotOptions](#)  
*Customize plots and figures.*
- class [teb::TebPlotter](#)  
*Provides useful visualization and plotting functions.*

### 14.4.1 Detailed Description

## Chapter 15

# Namespace Documentation

### 15.1 teb Namespace Reference

General project namespace for all library functions and objects.

#### Classes

- class [BaseController](#)  
*Base controller class (General Controller API)*
- class [BaseEdge](#)  
*Templated base edge class that stores an arbitrary number of values.*
- class [BaseEdge< D, FuncType >](#)  
*Templated base edge class that stores an arbitrary number of values (Partial template specialization that allows changing vertices dimensions at runtime).*
- class [EdgeSystemDynamics](#)  
*Equality constraint edge for satisfying continuous system dynamics specified by an [SystemDynamics](#) object.*
- class [BaseSolver](#)  
*Base class for solver implementations.*
- class [BaseSolverLeastSquares](#)  
*Extended base solver class for least squares optimizations.*
- struct [CustomBoundData](#)  
*Datastruct for custom bound data that can be queried from the [BoundConstraint](#) class.*
- class [BoundConstraint](#)  
*This class captures general bound constraints on optimization variables / vertices.*
- class [EdgeMinimizeTime](#)  
*Objective edge: minimize  $\Delta T$ .*
- class [EdgeQuadraticForm](#)  
*Objective edge: quadratic form for states and control input.*
- class [Config](#)  
*Configurations for Controller and Solver classes.*
- class [VertexType](#)  
*Generic interface class for vertices.*
- class [EdgeType](#)  
*Generic interface class for edges.*
- class [HyperGraph](#)  
*Graph object that stores pointers to active vertices and edges.*
- class [SystemDynamics](#)

- Helper class for modeling nonlinear dynamic systems.*

  - class [TebController](#)

*Main Timed-Elastic-Band controller class.*
  - class [StateVertex](#)

*Vertex that contains the StateVector.*
  - class [ControlVertex](#)

*Vertex that contains the ControlVector.*
  - class [TimeDiff](#)

*Vertex that contains the time information of the TEB:  $\Delta T$ .*
  - class [JacobianWorkspace](#)

*Memory management and accessor functions for the Block-Jacobians (for each edge).*
  - class [HessianWorkspace](#)

*Memory management and accessor functions for the Block-Hessian (for each edge).*
  - class [NumericalIntegrator](#)

*Interface for numerical integration methods used in simulation.*
  - class [ExplicitEuler](#)

*Simple explicit euler method for integration.*
  - class [RungeKuttaClassic](#)

*Classic Runge Kutta method (fourth order)*
  - class [RungeKutta5thOrder](#)

*Runge Kutta method (fifth order, slightly modified)*
  - class [SimResults](#)

*Simulation results are stored and combined in this container class.*
  - class [Simulator](#)

*[Simulator](#) for dynamic systems controlled by a [TebController](#).*
  - class [BaseSolverNonlinearProgramDense](#)

*Extended base solver class for nonlinear programs (dense version).*
  - class [SolverLevenbergMarquardtEigenDense](#)

*Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).*
  - class [SolverLevenbergMarquardtEigenSparse](#)

*Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).*
  - class [SolverNloptPackage](#)

*This class wraps the optimization problem to allow solving by NLOPT.*
  - class [SolverSQPDense](#)

*Dense sequential quadratic programming (SQP) solver for nonlinear programs.*
  - struct [PlotOptions](#)

*Customize plots and figures.*
  - class [TebPlotter](#)

*Provides useful visualization and plotting functions.*

## Typedefs

- using [EdgePositiveTime](#) = [BoundConstraint](#)< [BOUND\\_TYPE::LOWER](#), [TimeDiff](#), [BOUND\\_VARS::SINGLE](#), 0 >

*Bound constraint edge for the time difference:  $\Delta T \geq \varepsilon$ .*
- template<int q>
  - using [EdgeControlBounds](#) = [BoundConstraint](#)< [BOUND\\_TYPE::LOWERUPPER](#), [ControlVertex](#)< q >, [BOUND\\_VARS::ALL](#) >

*Bound constraint edge for control inputs:  $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$ .*

- `template<int p>`  
`using EdgeStateBounds = BoundConstraint< BOUND_TYPE::LOWERUPPER, StateVertex< p >, BOUND_VARS::ALL >`  
*Bound constraint edge for states:  $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$ .*
- `template<typename T >`  
`using EigenScalar = Eigen::Matrix< T, 1, 1 >`  
*Define 1x1 Eigen::Matrix.*
- `using EigenScalarD = EigenScalar< double >`  
*Define 1x1 Eigen::Matrix of type double.*
- `template<typename T >`  
`using BackupStackType = std::stack< T, std::deque< T > >`  
*Backup stack type.*

## Enumerations

- `enum BOUND_TYPE { BOUND_TYPE::LOWER, BOUND_TYPE::UPPER, BOUND_TYPE::LOWERUPPER }`  
*Enumerator class that stores all types of bounds for consideration in BoundConstraint class.*
- `enum BOUND_VARS { BOUND_VARS::ALL, BOUND_VARS::SINGLE }`  
*Enumerator class that stores all types of variables that can be bounded using BoundConstraint class.*
- `enum FiniteDifferences { FiniteDifferences::FORWARD, FiniteDifferences::CENTRAL }`  
*Enumeration for different finite difference types used especially for dynamic system discretization.*
- `enum HessianMethod { HessianMethod::NUMERIC, HessianMethod::BLOCK_BFGS, HessianMethod::FULL_BFGS, HessianMethod::FULL_BFGS_WITH_STRUCTURE_FILTER, HessianMethod::ZERO_HESSIAN }`  
*Enumeration for different hessian calculation methods (not necessary for least-squares solver)*
- `enum HessianInit { HessianInit::ZERO, HessianInit::IDENTITY, HessianInit::NUMERIC }`  
*Enumeration for the hessian initialization (relevant for BFGS hessian approximations)*
- `enum NloptAlgorithms { NloptAlgorithms::SLSQP }`  
*Enumeration for different solver algorithms supported by the Nlopt library (only those that are working here)*
- `enum FUNCT_TYPE { FUNCT_TYPE::NONLINEAR, FUNCT_TYPE::NONLINEAR_SQUARED, FUNCT_TYPE::NONLINEAR_ONCE_DIFF, FUNCT_TYPE::QUADRATIC, FUNCT_TYPE::LINEAR_SQUARED, FUNCT_TYPE::LINEAR }`  
*Enum for different function types (see EdgeType class)*
- `enum NumericalIntegrators { NumericalIntegrators::EXPLICIT_EULER, NumericalIntegrators::RUNGE_KUTTA_CLASSIC, NumericalIntegrators::RUNGE_KUTTA_5TH }`  
*Enumeration for different numerical integration methods.*

### 15.1.1 Typedef Documentation

15.1.1.1 `template<typename T > using teb::BackupStackType = typedef std::stack<T, std::deque<T> >`

Definition at line 52 of file typedefs.h.

15.1.1.2 `template<typename T > using teb::EigenScalar = typedef Eigen::Matrix<T,1,1>`

Definition at line 45 of file typedefs.h.

15.1.1.3 `using teb::EigenScalarD = typedef EigenScalar<double>`

Definition at line 48 of file typedefs.h.

## 15.1.2 Enumeration Type Documentation

15.1.2.1 `enum teb::BOUND_TYPE` [`strong`]

Enumerator

**LOWER**  
**UPPER**  
**LOWERUPPER**

Definition at line 14 of file bound\_constraints.h.

15.1.2.2 `enum teb::BOUND_VARS` [`strong`]

Enumerator

**ALL**  
**SINGLE**

Definition at line 16 of file bound\_constraints.h.

15.1.2.3 `enum teb::FiniteDifferences` [`strong`]

Enumerator

**FORWARD** Forward differences:  $\dot{x}(t) = \frac{x_{k+1} - x_k}{\Delta T}$ .  
**CENTRAL** Central differences:  $\dot{x}(t) = \frac{x_{k+1} - x_{k-1}}{2\Delta T}$ .

Definition at line 11 of file config.h.

15.1.2.4 `enum teb::FUNCT_TYPE` [`strong`]

Enumerator

**NONLINEAR** General nonlinear function that is at least two times differentiable.  
**NONLINEAR\_SQUARED** General nonlinear function that will be squared within the solver.  
**NONLINEAR\_ONCE\_DIFF** Nonlinear function that is only differentiable once: the hessian is zero.  
**QUADRATIC** Quadratic function (hessian is constant)  
**LINEAR\_SQUARED** Linear function that will be squared by the compiler.  
**LINEAR** Linear function (hessian is zero)

Definition at line 79 of file graph.h.

15.1.2.5 `enum teb::HessianInit` [`strong`]

Enumerator

**ZERO** Use the zero matrix for initialization.  
**IDENTITY** Use the identity matrix for hessian initialization (scale it via seperate config parameter)  
**NUMERIC** Calculate the initialization of the hessian numerically using the original problem.

Definition at line 29 of file config.h.



15.1.2.6 `enum teb::HessianMethod` `[strong]`

Enumerator

**NUMERIC** Calculate hessian numerically via difference quotients (based on graph structure)

**BLOCK\_BFGS** Calculate hessian for each block (defined by the graph) via BFGS.

**FULL\_BFGS** Calculate the hessian via BFGS quasi-netwon method without considering strucutre.

**FULL\_BFGS\_WITH\_STRUCTURE\_FILTER** Calculate hessian like with FULL\_BFGS but consider structure for non-zeros.

**ZERO\_HESSIAN** Set Hessian to zero.

Definition at line 19 of file config.h.

15.1.2.7 `enum teb::NloptAlgorithms` `[strong]`

Enumerator

**SLSQP** Sequential quadratic programming approach.

Definition at line 38 of file config.h.

15.1.2.8 `enum teb::NumericalIntegrators` `[strong]`

Enumerator

**EXPLICIT\_EULER** Forward Euler.

See Also

[ExplicitEuler](#)

**RUNGE\_KUTTA\_CLASSIC** Runge Kutta 4th Order.

See Also

[RungeKuttaClassic](#)

**RUNGE\_KUTTA\_5TH** Runge Kutta 5th Order.

See Also

[RungeKutta5thOrder](#)

Definition at line 12 of file integrators.h.



## Chapter 16

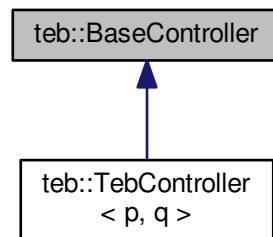
# Class Documentation

### 16.1 teb::BaseController Class Reference

Base controller class (General Controller API)

```
#include <base_controller.h>
```

Inheritance diagram for teb::BaseController:



#### Public Member Functions

- virtual double [getDt](#) () const =0  
*Get the time difference  $\Delta T$  between two discrete samples within the horizon.*
- virtual unsigned int [getN](#) () const =0  
*Get the horizon length, resp.*
- virtual void [step](#) (const double \*const x0, const double \*const xf, double \*ctrl\_out=nullptr)=0  
*Perform complete MPC step (initialization if necessary or update and optimization)*
- virtual Eigen::MatrixXd [getStateCtrlInfoMat](#) () const =0  
*Get a matrix containing all state and control input sequences.*
- virtual Eigen::VectorXd [getAbsoluteTimeVec](#) () const =0  
*Get the vector of absolute times for the complete horizon.*
- virtual Eigen::VectorXd [firstState](#) () const =0  
*Access the first state  $\mathbf{x}_0$  of the horizon.*
- virtual Eigen::VectorXd [lastState](#) () const =0

- virtual `Eigen::VectorXd firstControl () const =0`  
Access the last state  $\mathbf{x}_n$  of the horizon.
- virtual `Eigen::MatrixXd returnControlInputSequence () const =0`  
Access the first control input  $\mathbf{u}_1$  of the horizon.  
Return the complete control input sequence  $u_k, k = 1, \dots, n-1$ .
- virtual void `resetController ()=0`  
Reset the controller.

### 16.1.1 Detailed Description

This abstract class defines the general controller interface.

**Todo** Implement complete API (abstract functions, etc.) to allow a generic controller usage for `teb::Simulator` and `teb::TebPlotter`

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

`teb::TebController`

Definition at line 27 of file `base_controller.h`.

### 16.1.2 Member Function Documentation

16.1.2.1 virtual `Eigen::VectorXd teb::BaseController::firstControl ( ) const` [pure virtual]

#### Returns

Copy of the first state

Implemented in `teb::TebController< p, q >`.

Referenced by `teb::Simulator< p, q >::simClosedLoop()`.

16.1.2.2 virtual `Eigen::VectorXd teb::BaseController::firstState ( ) const` [pure virtual]

#### Returns

Copy of the first state

Implemented in `teb::TebController< p, q >`.

Referenced by `teb::Simulator< p, q >::simOpenLoop()`.

16.1.2.3 virtual `Eigen::VectorXd teb::BaseController::getAbsoluteTimeVec ( ) const` [pure virtual]

In case of a uniform step width / time difference (see `getDt()`) the vector corresponds to  $t = [0, \Delta T, 2\Delta T, \dots, n\Delta T]$ .

#### Returns

vector of absolute times starting with  $t=0$ . Get vector of absolute times  $t = [0, \Delta T, 2\Delta T, \dots, n\Delta T]$ .

Implemented in `teb::TebController< p, q >`.

Referenced by `teb::Simulator< p, q >::simOpenLoop()`.

16.1.2.4 `virtual double teb::BaseController::getDt ( ) const` [pure virtual]

It can also be interpreted as the step width.

Returns

$\Delta T$

Implemented in [teb::TebController< p, q >](#).

Referenced by [teb::Simulator< p, q >::simClosedLoop\(\)](#), and [teb::Simulator< p, q >::simOpenLoop\(\)](#).

16.1.2.5 `virtual unsigned int teb::BaseController::getN ( ) const` [pure virtual]

the number of discrete samples.

Returns

Number of discrete samples

Implemented in [teb::TebController< p, q >](#).

Referenced by [teb::Simulator< p, q >::simOpenLoop\(\)](#).

16.1.2.6 `virtual Eigen::MatrixXd teb::BaseController::getStateCtrlInfoMat ( ) const` [pure virtual]

The purpose of this function is mainly for debugging and visualization stuff.

Returns

[p+q x n] matrix with p states, q control inputs and n=[getN\(\)](#).

Implemented in [teb::TebController< p, q >](#).

Referenced by [teb::Simulator< p, q >::simOpenLoop\(\)](#).

16.1.2.7 `virtual Eigen::VectorXd teb::BaseController::lastState ( ) const` [pure virtual]

Returns

Copy of the last state

Implemented in [teb::TebController< p, q >](#).

16.1.2.8 `virtual void teb::BaseController::resetController ( )` [pure virtual]

Implemented in [teb::TebController< p, q >](#).

Referenced by [teb::Simulator< p, q >::simClosedLoop\(\)](#), and [teb::Simulator< p, q >::simOpenLoop\(\)](#).

16.1.2.9 `virtual Eigen::MatrixXd teb::BaseController::returnControlInputSequence ( ) const` [pure virtual]

Returns

Matrix containing all planned control inputs [q x n-1]

Implemented in [teb::TebController< p, q >](#).

Referenced by [teb::Simulator< p, q >::simOpenLoop\(\)](#).

16.1.2.10 `virtual void teb::BaseController::step ( const double *const x0, const double *const xf, double * ctrl_out = nullptr ) [pure virtual]`

## Parameters

<code>x0</code>	double array containing start state <code>x0</code> with <code>p</code> components [ <code>p</code> x 1]
<code>xf</code>	double array containing final state <code>xf</code> with <code>p</code> components [ <code>p</code> x 1]
<code>ctrl_out</code>	[output] store control input ( <a href="#">firstControl()</a> ) to control the plant [ <code>q</code> x 1]

Implemented in [teb::TebController< p, q >](#).

Referenced by `teb::Simulator< p, q >::simClosedLoop()`, and `teb::Simulator< p, q >::simOpenLoop()`.

The documentation for this class was generated from the following file:

- [base\\_controller.h](#)

## 16.2 `teb::BaseEdge< D, FuncType, Vertices >` Class Template Reference

Templated base edge class that stores an arbitrary number of values.

```
#include <base_edge.h>
```

Inheritance diagram for `teb::BaseEdge< D, FuncType, Vertices >`:



### Public Types

- using [ValueVector](#) = `Eigen::Matrix< double, D, 1 >`  
*Typedef to represent the static Eigen::Vector of cost/constraint values.*
- using [EdgeContainer](#) = `std::vector< EdgeType * >`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*
- using [ValueVectorMap](#) = `Eigen::Map< Eigen::VectorXd >`  
*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

### Public Member Functions

- virtual `int dimension () const`  
*Return edge dimension by function call.*
- `BaseEdge ()=delete`  
*Delete default constructor.*
- `template<class... VerticesT> BaseEdge (VerticesT &...args)`
- virtual `~BaseEdge ()`  
*< Construct edge by passing all vertices references.*
- virtual `FUNCT_TYPE functType () const`  
*Return function typed given by template parameter FuncType.*
- virtual `const double * valuesData () const`  
*Return pointer to cost/constraint values [`dimension()` x 1] (read-only).*

- virtual double \* [valuesData](#) ()  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1].*
- virtual const [ValueVector](#) & [values](#) () const  
*Return cost/constraint values as static [Eigen::Vector](#) (read-only).*
- virtual [ValueVector](#) & [values](#) ()  
*Return cost/constraint values as static [Eigen::Vector](#).*
- virtual void [allocateMemory](#) (bool skip\_hessian=false)  
*Allocate memory for [JacobianWorkspace](#) and [HessianWorkspace](#) and get vertices dimensions.*
- virtual [VertexType](#) \* [getVertex](#) (unsigned int idx)  
*Access attached vertex with index [idx](#).*
- virtual const [VertexType](#) \* [getVertex](#) (unsigned int idx) const  
*Access attached vertex with index [idx](#) (read-only).*
- virtual unsigned int [noVertices](#) () const  
*Return the number of attached vertices.*
- virtual void [calculateVertexDimensions](#) ()  
*Return the number of attached vertices.*
- virtual const std::array  
< [VertexType](#) \*, [NoVertices](#) > & [vertices](#) () const  
*Return vertex container of the edge.*
- virtual void [computeValues](#) ()  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual void [computeJacobian](#) ()  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual void [computeHessian](#) ()  
*Compute Block-Hessian and store it to [EdgeType::\\_hessians](#) according to the description in [HessianWorkspace](#).*
- virtual const bool [isBoundConstraint](#) () const  
*Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.*
- virtual [ValueVectorMap](#) [valuesMap](#) ()  
*Get cost/constraints values as [Eigen](#) type matrix.*
- virtual void \* [getCustomData](#) ()  
*Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.*
- virtual [JacobianWorkspace](#) & [jacobians](#) ()  
*Access the jacobian workspace of this edge.*
- virtual const [JacobianWorkspace](#) & [jacobians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual [HessianWorkspace](#) & [hessians](#) ()  
*Access the hessian workspace of this edge.*
- virtual const [HessianWorkspace](#) & [hessians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual void [backupJacobian](#) ()  
*Make a backup of the current jacobian block matrix.*
- virtual void [restoreJacobian](#) ()  
*Restore jacobian block matrix from the backup stack.*
- virtual void [discardJacobianBackup](#) ()  
*Discard current jacobian backup.*
- virtual [JacobianWorkspace](#) & [getJacobianBackup](#) ()



## Static Public Attributes

- static constexpr const int `NoVertices` = sizeof...(Vertices)
- static const int `Dimension` = D

*Edge dimension as static member variable.*

## Protected Attributes

- `ValueVector _values` = `ValueVector::Zero()`  
*Actual cost/constraint value data object (initialized to zero).*
- const std::array< `VertexType` \*, `NoVertices` > `_vertices`  
*Container that stores all attached vertices.*
- std::array< int, `NoVertices` > `_vertices_dim`  
*Store dimensions for all vertices (`EdgeType::_vertices`, `calculateVertexDimensions()`)*
- `JacobianWorkspace _jacobians`  
*Block-Jacobians of the edge (see `JacobianWorkspace`).*
- `HessianWorkspace _hessians`  
*Block-Hessians of the edge (see `HessianWorkspace`).*
- `BackupStackType` < `JacobianWorkspace` > `_jacob_backup`

### 16.2.1 Detailed Description

```
template<int D, FUNCT_TYPE FuncType, class... Vertices>class teb::BaseEdge< D, FuncType, Vertices >
```

This class defines the basis for nearly all edges. The dimension has to be known at compile time and is set by the template parameter `D`.

#### Remarks

Call `allocateMemory()` before using the edge for optimization.

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

`EdgeType`

#### Template Parameters

<code>D</code>	Dimension of the edge (value vector).
<code>FuncType</code>	Function type of the underlying cost function (according to <code>FUNCT_TYPE</code> enum)
<code>Vertices...</code>	An arbitrary number of vertex types that are attached to this edge

Definition at line 41 of file `base_edge.h`.

### 16.2.2 Member Typedef Documentation

16.2.2.1 using `teb::EdgeType::EdgeContainer` = `std::vector<EdgeType*>` [inherited]

Definition at line 114 of file `graph.h`.

16.2.2.2 `template<int D, FUNCT_TYPE FuncType, class... Vertices> using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1>`

Definition at line 54 of file base\_edge.h.

16.2.2.3 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]`

Definition at line 116 of file graph.h.

## 16.2.3 Constructor & Destructor Documentation

16.2.3.1 `template<int D, FUNCT_TYPE FuncType, class... Vertices> teb::BaseEdge< D, FuncType, Vertices >::BaseEdge ( ) [delete]`

16.2.3.2 `template<int D, FUNCT_TYPE FuncType, class... Vertices> template<class... VerticesT> teb::BaseEdge< D, FuncType, Vertices >::BaseEdge ( VerticesT &... args ) [inline]`

Definition at line 60 of file base\_edge.h.

16.2.3.3 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual teb::BaseEdge< D, FuncType, Vertices >::~BaseEdge ( ) [inline],[virtual]`

Empty Destructor

Definition at line 66 of file base\_edge.h.

## 16.2.4 Member Function Documentation

16.2.4.1 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory ( bool skip_hessian = false ) [inline],[virtual]`

### Remarks

Before calling, call `resizeVertexContainer()` edge and add all vertices (`setVertex()`) first.

**Todo** Implement automatic memory allocation instead calling this function manually.

### Parameters

<i>skip_hessian</i>	If no hessian calculation is required by the solver skip memory allocation.
---------------------	---

Implements [teb::EdgeType](#).

Definition at line 84 of file base\_edge.h.

16.2.4.2 `virtual void teb::EdgeType::backupJacobian ( ) [inline],[virtual],[inherited]`

Definition at line 162 of file graph.h.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

16.2.4.3 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline],[virtual]`

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container `EdgeType::_vertices_dim`. This function is called within `BaseEdge::allocateMemory()`.

**Todo** Maybe implement static/constexpr version of collecting all dimensions since they are known at compile-time (maybe <http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array>)

Definition at line 110 of file `base_edge.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, and `teb::BaseEdge< D, FuncType >::allocateMemory()`.

**16.2.4.4** `template<int D, FUNCT_TYPE FuncType, class... Vertices> void teb::BaseEdge< D, FuncType, Vertices >::computeHessian ( ) [virtual]`

This implementation calculates the Block-Hessians for all statically allocated vertices and cost/constraint values numerically using forward differences.

For more information about the Hessian structure refer to [HessianWorkspace](#).

If the `FuncType` template parameter is set to `FUNCT_TYPE::LINEAR` or `FuncType == FUNCT_TYPE::NONLINEAR_ONCE_DIFF` the hessian is set to zero.

The results are stored to the class member `BaseEdgeBinary::_hessians`.

Feel free to reimplement this function in child classes in order to provide analytical Hessians.

#### Remarks

This function assumes that `computeJacobian()` was called before! It is used as reference for calculating the forward-differences.

#### See Also

[computeJacobian\(\)](#), [HessianWorkspace](#)

Implements `teb::EdgeType`.

Reimplemented in `teb::EdgeQuadraticForm< p, q >`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >`, and `teb::EdgeMinimizeTime`.

Definition at line 165 of file `base_edge.hpp`.

References `teb::VertexType::dimension()`, `teb::JacobianWorkspace::getWorkspace()`, `teb::VertexType::isFixedAll()`, `teb::LINEAR`, `teb::LINEAR_SQUARED`, `teb::NONLINEAR_ONCE_DIFF`, `teb::VertexType::plus()`, `teb::VertexType::pop()`, and `teb::VertexType::push()`.

**16.2.4.5** `template<int D, FUNCT_TYPE FuncType, class... Vertices> void teb::BaseEdge< D, FuncType, Vertices >::computeJacobian ( ) [virtual]`

This implementation calculates the Block-Jacobians for all statically allocated vertices numerically using central differences.

Compute Block-Jacobian and store it to `EdgeType::_jacobians` according to the description in [JacobianWorkspace](#).

For more information about the Jacobian structure refer to [JacobianWorkspace](#).

The results are stored to the class member `BaseEdgeBinary::_jacobians`.

The implementation is similar to [2].

Feel free to reimplement this function in child classes in order to provide analytical Jacobians.

See Also

[computeHessian\(\)](#), [JacobianWorkspace](#)

Implements [teb::EdgeType](#).

Reimplemented in [teb::EdgeQuadraticForm< p, q >](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#), and [teb::EdgeMinimizeTime](#).

Definition at line 24 of file `base_edge.hpp`.

References [teb::VertexType::dimension\(\)](#), [teb::VertexType::isFixedAll\(\)](#), [teb::VertexType::plus\(\)](#), [teb::VertexType::pop\(\)](#), and [teb::VertexType::push\(\)](#).

Referenced by [teb::BaseEdge< D, FuncType >::computeHessian\(\)](#).

```
16.2.4.6  template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices
>::computeValues ( ) [inline],[virtual]
```

Implements [teb::EdgeType](#).

Reimplemented in [teb::EdgeQuadraticForm< p, q >](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#), [teb::EdgeSystemDynamics< p, q, central >](#), and [teb::EdgeMinimizeTime](#).

Definition at line 125 of file `base_edge.h`.

Referenced by [teb::BaseEdge< D, FuncType >::computeJacobian\(\)](#).

```
16.2.4.7  template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual int teb::BaseEdge< D, FuncType, Vertices
>::dimension ( ) const [inline],[virtual]
```

Implements [teb::EdgeType](#).

Definition at line 51 of file `base_edge.h`.

```
16.2.4.8  virtual void teb::EdgeType::discardJacobianBackup ( ) [inline],[virtual],[inherited]
```

Definition at line 171 of file `graph.h`.

References [teb::EdgeType::\\_jacob\\_backup](#).

```
16.2.4.9  template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual FUNCT_TYPE teb::BaseEdge< D, FuncType,
Vertices >::funcType ( ) const [inline],[virtual]
```

Implements [teb::EdgeType](#).

Definition at line 69 of file `base_edge.h`.

```
16.2.4.10 virtual void* teb::EdgeType::getCustomData ( ) [inline],[virtual],[inherited]
```

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 154 of file `graph.h`.

```
16.2.4.11 virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline],[virtual],
[inherited]
```

Definition at line 173 of file `graph.h`.

References [teb::EdgeType::\\_jacob\\_backup](#).

16.2.4.12 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 98 of file `base_edge.h`.

16.2.4.13 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) const [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 99 of file `base_edge.h`.

16.2.4.14 `virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]`

Definition at line 159 of file `graph.h`.

References `teb::EdgeType::_hessians`.

16.2.4.15 `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]`

Definition at line 160 of file `graph.h`.

References `teb::EdgeType::_hessians`.

16.2.4.16 `virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 124 of file `graph.h`.

16.2.4.17 `virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]`

Definition at line 157 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

16.2.4.18 `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]`

Definition at line 158 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

16.2.4.19 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 101 of file `base_edge.h`.

16.2.4.20 `virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]`

Definition at line 165 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.2.4.21** `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline],[virtual]`

Definition at line 74 of file `base_edge.h`.

**16.2.4.22** `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline],[virtual]`

Definition at line 75 of file `base_edge.h`.

**16.2.4.23** `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline],[virtual]`

Implements [teb::EdgeType](#).

Definition at line 72 of file `base_edge.h`.

**16.2.4.24** `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline],[virtual]`

Implements [teb::EdgeType](#).

Definition at line 73 of file `base_edge.h`.

**16.2.4.25** `virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline],[virtual],[inherited]`

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file `graph.h`.

References `teb::EdgeType::dimension()`, and `teb::EdgeType::valuesData()`.

**16.2.4.26** `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const std::array<VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline],[virtual]`

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file `base_edge.h`.

## 16.2.5 Member Data Documentation

**16.2.5.1 HessianWorkspace** `teb::EdgeType::_hessians [protected],[inherited]`

Definition at line 178 of file `graph.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::EdgeMinimizeTime::computeHessian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian()`, `teb::EdgeQuadraticForm< p, q >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, and `teb::EdgeType::hessians()`.

### 16.2.5.2 `BackupStackType<JacobianWorkspace>` `teb::EdgeType::_jacob_backup` `[protected]`, `[inherited]`

Definition at line 180 of file `graph.h`.

Referenced by `teb::EdgeType::backupJacobian()`, `teb::EdgeType::discardJacobianBackup()`, `teb::EdgeType::getJacobianBackup()`, and `teb::EdgeType::restoreJacobian()`.

### 16.2.5.3 `JacobianWorkspace` `teb::EdgeType::_jacobians` `[protected]`, `[inherited]`

Definition at line 173 of file `graph.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::EdgeType::backupJacobian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::EdgeMinimizeTime::computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian()`, `teb::EdgeQuadraticForm< p, q >::computeJacobian()`, `teb::BaseEdge< D, FuncType >::computeJacobian()`, `teb::EdgeType::jacobians()`, and `teb::EdgeType::restoreJacobian()`.

### 16.2.5.4 `template<int D, FUNCT_TYPE FuncType, class... Vertices> ValueVector` `teb::BaseEdge< D, FuncType, Vertices >::_values = ValueVector::Zero()` `[protected]`

Definition at line 139 of file `base_edge.h`.

Referenced by `teb::BaseEdge< D, FuncType >::computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues()`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::values()`, `teb::BaseEdge< D, FuncType >::values()`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::valuesData()`, and `teb::BaseEdge< D, FuncType >::valuesData()`.

### 16.2.5.5 `template<int D, FUNCT_TYPE FuncType, class... Vertices> const std::array<VertexType*, NoVertices>` `teb::BaseEdge< D, FuncType, Vertices >::_vertices` `[protected]`

Definition at line 141 of file `base_edge.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions()`, `teb::BaseEdge< D, FuncType >::calculateVertexDimensions()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData()`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::getVertex()`, `teb::BaseEdge< D, FuncType >::getVertex()`, `teb::BaseEdge< D, FuncType >::noVertices()`, `teb::BaseEdge< D, FuncType >::resizeVertexContainer()`, `teb::BaseEdge< D, FuncType >::setVertex()`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::vertices()`, and `teb::BaseEdge< D, FuncType >::vertices()`.

### 16.2.5.6 `template<int D, FUNCT_TYPE FuncType, class... Vertices> std::array<int, NoVertices>` `teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim` `[protected]`

Definition at line 142 of file `base_edge.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions()`, and `teb::BaseEdge< D, FuncType >::calculateVertexDimensions()`.

### 16.2.5.7 `template<int D, FUNCT_TYPE FuncType, class... Vertices> const int` `teb::BaseEdge< D, FuncType, Vertices >::Dimension = D` `[static]`

Definition at line 50 of file `base_edge.h`.

16.2.5.8 `template<int D, FUNCT_TYPE FuncType, class... Vertices> constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices = sizeof...(Vertices) [static]`

Definition at line 48 of file `base_edge.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::noVertices()`.

The documentation for this class was generated from the following files:

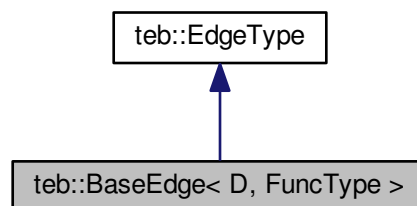
- [base\\_edge.h](#)
- [base\\_edge.hpp](#)

## 16.3 `teb::BaseEdge< D, FuncType >` Class Template Reference

Templated base edge class that stores an arbitrary number of values (Partial template specialization that allows changing vertices dimensions at runtime).

```
#include <base_edge.h>
```

Inheritance diagram for `teb::BaseEdge< D, FuncType >`:



### Public Types

- using `ValueVector` = `Eigen::Matrix< double, D, 1 >`  
*Typedef to represent the static Eigen::Vector of cost/constraint values.*
- using `EdgeContainer` = `std::vector< EdgeType * >`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*
- using `ValueVectorMap` = `Eigen::Map< Eigen::VectorXd >`  
*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

### Public Member Functions

- virtual `int dimension () const`  
*Return edge dimension by function call.*
- `BaseEdge ()`  
*Empty Constructor.*
- virtual `~BaseEdge ()`  
*Empty Destructor.*
- virtual `FUNCT_TYPE funcType () const`  
*Return function typed given by template parameter FuncType.*



- virtual const double \* [valuesData](#) () const  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1] (read-only).*
- virtual double \* [valuesData](#) ()  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1].*
- virtual const [ValueVector](#) & [values](#) () const  
*Return cost/constraint values as static Eigen::Vector (read-only).*
- virtual [ValueVector](#) & [values](#) ()  
*Return cost/constraint values as static Eigen::Vector.*
- virtual void [allocateMemory](#) (bool skip\_hessian=false)  
*Allocate memory for [JacobianWorkspace](#) and [HessianWorkspace](#) and get vertices dimensions.*
- virtual [VertexType](#) \* [getVertex](#) (unsigned int idx)  
*Access attached vertex with index [idx](#).*
- virtual const [VertexType](#) \* [getVertex](#) (unsigned int idx) const  
*Access attached vertex with index [idx](#) (read-only).*
- virtual unsigned int [noVertices](#) () const  
*Return the number of attached vertices.*
- virtual void [calculateVertexDimensions](#) ()  
*Return the number of attached vertices.*
- virtual void [setVertex](#) (unsigned int idx, [VertexType](#) \*pvertex)  
*Attach a new vertex of type [TebVertex\\*](#) with index [idx](#) to the edge.*
- virtual const std::vector  
< [VertexType](#) \* > & [vertices](#) () const  
*Return vertex container of the edge.*
- virtual std::vector  
< [VertexType](#) \* > & [vertices](#) ()  
*Return vertex container of the edge.*
- virtual void [computeJacobian](#) ()  
*Compute Block-Jacobian and store it to [EdgeType::\\_jacobians](#) according to the description in [JacobianWorkspace](#).*
- virtual void [computeHessian](#) ()  
*Compute Block-Hessian and store it to [EdgeType::\\_hessians](#) according to the description in [HessianWorkspace](#).*
- virtual const bool [isBoundConstraint](#) () const  
*Override this method in a subclass that constitutes a bound constraint since it can be handled separately by some solvers to speed up optimization.*
- virtual void [computeValues](#) ()=0  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual [ValueVectorMap](#) [valuesMap](#) ()  
*Get cost/constraints values as Eigen type matrix.*
- virtual void \* [getCustomData](#) ()  
*Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.*
- virtual [JacobianWorkspace](#) & [jacobians](#) ()  
*Access the jacobian workspace of this edge.*
- virtual const [JacobianWorkspace](#) & [jacobians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual [HessianWorkspace](#) & [hessians](#) ()  
*Access the hessian workspace of this edge.*
- virtual const [HessianWorkspace](#) & [hessians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual void [backupJacobian](#) ()  
*Make a backup of the current jacobian block matrix.*
- virtual void [restoreJacobian](#) ()

- *Restore jacobian block matrix from the backup stack.*
- virtual void [discardJacobianBackup](#) ()
- *Discard current jacobian backup.*
- virtual [JacobianWorkspace](#) & [getJacobianBackup](#) ()

### Static Public Attributes

- static const int [Dimension](#) = D
- *Edge dimension as static member variable.*

### Protected Member Functions

- void [resizeVertexContainer](#) (unsigned int n)
- *Set number  $n$  of vertices attached to this edge.*

### Protected Attributes

- [ValueVector](#) [\\_values](#) = ValueVector::Zero()
- *Actual cost/constraint value data object (initialized to zero).*
- std::vector< [VertexType](#) \* > [\\_vertices](#)
- *Container that stores all attached vertices.*
- std::vector< int > [\\_vertices\\_dim](#)
- *Store dimensions for all vertices (EdgeType::\_vertices, [calculateVertexDimensions\(\)](#))*
- [JacobianWorkspace](#) [\\_jacobians](#)
- *Block-Jacobians of the edge (see [JacobianWorkspace](#)).*
- [HessianWorkspace](#) [\\_hessians](#)
- *Block-Hessians of the edge (see [HessianWorkspace](#))*
- [BackupStackType](#)
- [< JacobianWorkspace > \\_jacob\\_backup](#)

## 16.3.1 Detailed Description

```
template<int D, FUNCT_TYPE FuncType>class teb::BaseEdge< D, FuncType >
```

This class defines the basis for nearly all edges. The dimension has to be known at compile time and is set by the template parameter `D`.

#### Remarks

Call [allocateMemory\(\)](#) before using the edge for optimization.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### See Also

[EdgeType](#)

## Template Parameters

<i>D</i>	Dimension of the edge (value vector).
<i>FuncType</i>	Function type of the underlying cost function (according to <code>FUNC_TYPE</code> enum)

Definition at line 172 of file `base_edge.h`.

## 16.3.2 Member Typedef Documentation

16.3.2.1 `using teb::EdgeType::EdgeContainer = std::vector<EdgeType*>` `[inherited]`

Definition at line 114 of file `graph.h`.

16.3.2.2 `template<int D, FUNCT_TYPE FuncType> using teb::BaseEdge< D, FuncType >::ValueVector = Eigen::Matrix<double, D, 1>`

Definition at line 180 of file `base_edge.h`.

16.3.2.3 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd >` `[inherited]`

Definition at line 116 of file `graph.h`.

## 16.3.3 Constructor &amp; Destructor Documentation

16.3.3.1 `template<int D, FUNCT_TYPE FuncType> teb::BaseEdge< D, FuncType >::BaseEdge ( )` `[inline]`

Definition at line 183 of file `base_edge.h`.

16.3.3.2 `template<int D, FUNCT_TYPE FuncType> virtual teb::BaseEdge< D, FuncType >::~BaseEdge ( )`  
`[inline], [virtual]`

Definition at line 184 of file `base_edge.h`.

## 16.3.4 Member Function Documentation

16.3.4.1 `template<int D, FUNCT_TYPE FuncType> virtual void teb::BaseEdge< D, FuncType >::allocateMemory ( bool skip_hessian = false )` `[inline], [virtual]`

## Remarks

Before calling, call [resizeVertexContainer\(\)](#) edge and add all vertices ([setVertex\(\)](#)) first.

**Todo** Implement automatic memory allocation instead calling this function manually.

## Parameters

<i>skip_hessian</i>	If no hessian calculation is required by the solver skip memory allocation.
---------------------	---

Implements [teb::EdgeType](#).

Definition at line 202 of file `base_edge.h`.

References [teb::EdgeType::\\_hessians](#), [teb::EdgeType::\\_jacobians](#), [teb::BaseEdge< D, FuncType, Vertices >::\\_vertices\\_dim](#), [teb::JacobianWorkspace::allocate\(\)](#), [teb::HessianWorkspace::allocate\(\)](#), and [teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions\(\)](#).

**16.3.4.2** `virtual void teb::EdgeType::backupJacobian ( ) [inline],[virtual],[inherited]`

Definition at line 162 of file graph.h.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.3.4.3** `template<int D, FUNCT_TYPE FuncType> virtual void teb::BaseEdge< D, FuncType >::calculateVertexDimensions ( ) [inline],[virtual]`

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container `EdgeType::_vertices_dim`. This function is called within [BaseEdge::allocateMemory\(\)](#).

**Todo** Maybe implement another strategy to store the dimensions.

Definition at line 227 of file base\_edge.h.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`, and `teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim`.

**16.3.4.4** `template<int D, FUNCT_TYPE FuncType> void teb::BaseEdge< D, FuncType >::computeHessian ( ) [virtual]`

This implementation calculates the Block-Hessians for all dynamically allocated vertices and cost/constraint values numerically using forward differences.

For more information about the Hessian structure refer to [HessianWorkspace](#).

If the `FuncType` template parameter is set to [FUNCT\\_TYPE::LINEAR](#) or `FuncType == FUNCT_TYPE::NONLINEAR_ONCE_DIFF` the hessian is set to zero.

The results are stored to the class member `BaseEdgeBinary::_hessians`.

Feel free to reimplement this function in child classes in order to provide analytical Hessians.

#### Remarks

This function assumes that [computeJacobian\(\)](#) was called before! It is used as reference for calculating the forward-differences.

#### See Also

[computeJacobian\(\)](#), [HessianWorkspace](#)

Implements [teb::EdgeType](#).

Definition at line 277 of file base\_edge.hpp.

References `teb::EdgeType::_hessians`, `teb::EdgeType::_jacobians`, `teb::BaseEdge< D, FuncType, Vertices >::_vertices`, `teb::BaseEdge< D, FuncType, Vertices >::computeJacobian()`, `teb::VertexType::dimension()`, `teb::JacobianWorkspace::getWorkspace()`, `teb::HessianWorkspace::getWorkspace()`, `teb::VertexType::isFixedAll()`, `teb::LINEAR`, `teb::LINEAR_SQUARED`, `teb::NONLINEAR_ONCE_DIFF`, `teb::VertexType::plus()`, `teb::VertexType::pop()`, and `teb::VertexType::push()`.

**16.3.4.5** `template<int D, FUNCT_TYPE FuncType> void teb::BaseEdge< D, FuncType >::computeJacobian ( ) [virtual]`

This implementation calculates the Block-Jacobians for all dynamically allocated vertices numerically using central differences.

For more information about the Jacobian structure refer to [JacobianWorkspace](#).

The results are stored to the class member `BaseEdgeBinary::_jacobians`.

The implementation is similar to [2].

Feel free to reimplement this function in child classes in order to provide analytical Jacobians.

See Also

[computeHessian\(\)](#), [JacobianWorkspace](#)

Implements [teb::EdgeType](#).

Definition at line 92 of file `base_edge.hpp`.

References `teb::EdgeType::_jacobians`, `teb::BaseEdge< D, FuncType, Vertices >::_values`, `teb::BaseEdge< D, FuncType, Vertices >::_vertices`, `teb::BaseEdge< D, FuncType, Vertices >::computeValues()`, `teb::VertexType::dimension()`, `teb::JacobianWorkspace::getWorkspace()`, `teb::VertexType::isFixedAll()`, `teb::VertexType::plus()`, `teb::VertexType::pop()`, and `teb::VertexType::push()`.

**16.3.4.6** `virtual void teb::EdgeType::computeValues ( ) [pure virtual],[inherited]`

Implemented in [teb::EdgeQuadraticForm< p, q >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#), [teb::EdgeSystemDynamics< p, q, central >](#), and [teb::EdgeMinimizeTime](#).

**16.3.4.7** `template<int D, FUNCT_TYPE FuncType> virtual int teb::BaseEdge< D, FuncType >::dimension ( ) const [inline],[virtual]`

Implements [teb::EdgeType](#).

Definition at line 177 of file `base_edge.h`.

**16.3.4.8** `virtual void teb::EdgeType::discardJacobianBackup ( ) [inline],[virtual],[inherited]`

Definition at line 171 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

**16.3.4.9** `template<int D, FUNCT_TYPE FuncType> virtual FUNCT_TYPE teb::BaseEdge< D, FuncType >::funcType ( ) const [inline],[virtual]`

Implements [teb::EdgeType](#).

Definition at line 187 of file `base_edge.h`.

**16.3.4.10** `virtual void* teb::EdgeType::getCustomData ( ) [inline],[virtual],[inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 154 of file `graph.h`.

**16.3.4.11** `virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline],[virtual],[inherited]`

Definition at line 173 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

**16.3.4.12** `template<int D, FUNCT_TYPE FuncType> virtual VertexType* teb::BaseEdge< D, FuncType >::getVertex ( unsigned int idx ) [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 215 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.3.4.13** `template<int D, FUNCT_TYPE FuncType> virtual const VertexType* teb::BaseEdge< D, FuncType >::getVertex ( unsigned int idx ) const [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 216 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.3.4.14** `virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline], [virtual], [inherited]`

Definition at line 159 of file `graph.h`.

References `teb::EdgeType::_hessians`.

**16.3.4.15** `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline], [virtual], [inherited]`

Definition at line 160 of file `graph.h`.

References `teb::EdgeType::_hessians`.

**16.3.4.16** `virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline], [virtual], [inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 124 of file `graph.h`.

**16.3.4.17** `virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline], [virtual], [inherited]`

Definition at line 157 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

**16.3.4.18** `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline], [virtual], [inherited]`

Definition at line 158 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

**16.3.4.19** `template<int D, FUNCT_TYPE FuncType> virtual unsigned int teb::BaseEdge< D, FuncType >::noVertices ( ) const [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 218 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.3.4.20** `template<int D, FUNCT_TYPE FuncType> void teb::BaseEdge< D, FuncType >::resizeVertexContainer ( unsigned int n ) [inline], [protected]`

Definition at line 282 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.3.4.21** `virtual void teb::EdgeType::restoreJacobian ( ) [inline], [virtual], [inherited]`

Definition at line 165 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.3.4.22** `template<int D, FUNCT_TYPE FuncType> virtual void teb::BaseEdge< D, FuncType >::setVertex ( unsigned int idx, VertexType * pvertex ) [inline], [virtual]`

Make sure that memory is allocated before using `resizeVertexContainer()` or that it is controlled by a specific child class.

Parameters

<i>idx</i>	index of the vertex inside this edge. Start with 0.
<i>pvertex</i>	pointer to the <code>TebVertex</code> object or to one of its children.

Definition at line 246 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.3.4.23** `template<int D, FUNCT_TYPE FuncType> virtual const ValueVector& teb::BaseEdge< D, FuncType >::values ( ) const [inline], [virtual]`

Definition at line 192 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.3.4.24** `template<int D, FUNCT_TYPE FuncType> virtual ValueVector& teb::BaseEdge< D, FuncType >::values ( ) [inline], [virtual]`

Definition at line 193 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.3.4.25** `template<int D, FUNCT_TYPE FuncType> virtual const double* teb::BaseEdge< D, FuncType >::valuesData ( ) const [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 190 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.3.4.26** `template<int D, FUNCT_TYPE FuncType> virtual double* teb::BaseEdge< D, FuncType >::valuesData ( ) [inline], [virtual]`

Implements [teb::EdgeType](#).

Definition at line 191 of file base\_edge.h.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.3.4.27** `virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline],[virtual],[inherited]`

return `Eigen::Vector` type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References `teb::EdgeType::dimension()`, and `teb::EdgeType::valuesData()`.

**16.3.4.28** `template<int D, FUNCT_TYPE FuncType> virtual const std::vector<VertexType*>& teb::BaseEdge< D, FuncType >::vertices ( ) const [inline],[virtual]`

Returns

(Read-only) reference to the vertex container.

Definition at line 256 of file base\_edge.h.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.3.4.29** `template<int D, FUNCT_TYPE FuncType> virtual std::vector<VertexType*>& teb::BaseEdge< D, FuncType >::vertices ( ) [inline],[virtual]`

Returns

Reference to the vertex container.

Definition at line 262 of file base\_edge.h.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

## 16.3.5 Member Data Documentation

**16.3.5.1** `HessianWorkspace teb::EdgeType::_hessians [protected],[inherited]`

Definition at line 178 of file graph.h.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `allocateMemory()`, `teb::EdgeMinimizeTime::computeHessian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian()`, `teb::EdgeQuadraticForm< p, q >::computeHessian()`, `computeHessian()`, and `teb::EdgeType::_hessians()`.

**16.3.5.2** `BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected],[inherited]`

Definition at line 180 of file graph.h.

Referenced by `teb::EdgeType::backupJacobian()`, `teb::EdgeType::discardJacobianBackup()`, `teb::EdgeType::getJacobianBackup()`, and `teb::EdgeType::restoreJacobian()`.

**16.3.5.3** `JacobianWorkspace teb::EdgeType::_jacobians [protected],[inherited]`

Definition at line 173 of file graph.h.



Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `allocateMemory()`, `teb::EdgeType::backupJacobian()`, `computeHessian()`, `teb::EdgeMinimizeTime::computeJacobian()`, `teb::Bound-Constraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian()`, `teb::EdgeQuadraticForm< p, q >::computeJacobian()`, `computeJacobian()`, `teb::EdgeType::jacobians()`, and `teb::EdgeType::restoreJacobian()`.

**16.3.5.4** `template<int D, FUNCT_TYPE FuncType> ValueVector teb::BaseEdge< D, FuncType >::_values = ValueVector::Zero()` [protected]

Definition at line 276 of file `base_edge.h`.

**16.3.5.5** `template<int D, FUNCT_TYPE FuncType> std::vector<VertexType*> teb::BaseEdge< D, FuncType >::_vertices` [protected]

Definition at line 278 of file `base_edge.h`.

**16.3.5.6** `template<int D, FUNCT_TYPE FuncType> std::vector<int> teb::BaseEdge< D, FuncType >::_vertices_dim` [protected]

Definition at line 279 of file `base_edge.h`.

**16.3.5.7** `template<int D, FUNCT_TYPE FuncType> const int teb::BaseEdge< D, FuncType >::Dimension = D` [static]

Definition at line 176 of file `base_edge.h`.

The documentation for this class was generated from the following files:

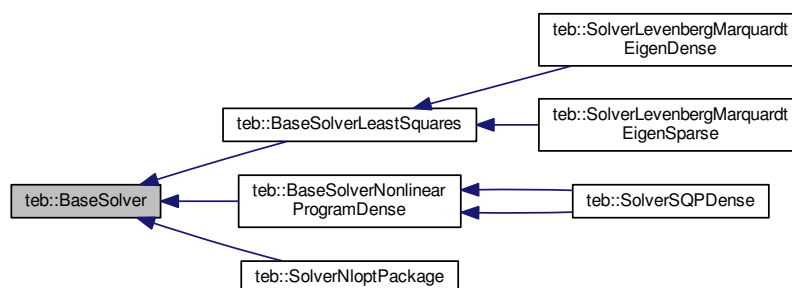
- [base\\_edge.h](#)
- [base\\_edge.hpp](#)

## 16.4 teb::BaseSolver Class Reference

Base class for solver implementations.

```
#include <base_solver.h>
```

Inheritance diagram for `teb::BaseSolver`:



## Public Types

- using [VertexContainer](#) = [HyperGraph::VertexContainer](#)  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using [EdgeContainer](#) = [HyperGraph::EdgeContainer](#)  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

## Public Member Functions

- [BaseSolver](#) ()  
*Empty Constructor.*
- virtual [~BaseSolver](#) ()  
*Empty Destructor.*
- void [setConfig](#) (const [Config](#) \*config)  
*Set config including solver settings.*
- virtual bool [solve](#) ([HyperGraph](#) \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

## Public Attributes

- const [Config](#) \* [cfg](#) = nullptr  
*Store pointer to [Config](#) object.*
- [EIGEN\\_MAKE\\_ALIGNED\\_OPERATOR\\_NEW](#)

## Protected Member Functions

- virtual bool [solveImpl](#) ()=0  
*Solve the actual optimization problem.*
- virtual void [initWorkspaces](#) ()  
*Initialize workspaces for the block jacobians and hessians.*
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const  
*Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- `VertexContainer * _active_vertices = nullptr`  
*Pointer to active vertex container (active = non-fixed)*
- `EdgeContainer * _objectives = nullptr`  
*Pointer to edges representing objective functions.*
- `EdgeContainer * _equalities = nullptr`  
*Pointer to edges representing equality constraints.*
- `EdgeContainer * _inequalities = nullptr`  
*Pointer to edges representing inequality constraints.*
- `int _opt_vec_dim = -1`  
*Store dimension of the optimization vector here (see [getOptVecDimension\(\)](#)).*
- `int _objective_dim = -1`  
*Store dimension of the objective function (see [getDimObjectives\(\)](#)).*
- `int _equalities_dim = -1`  
*Store dimension of the equality constraints (see [getDimEqualities\(\)](#)).*
- `int _inequalities_dim = -1`  
*Store dimension of the inequality constraints (see [getDimInequalities\(\)](#)).*
- `unsigned int _no_vert_backups = 0`  
*Track number of vertices backups made.*
- `bool _graph_structure_modified = true`  
*Mark if a new graph structure is available, thus we need to reinitialize all workspaces.*

### 16.4.1 Detailed Description

This abstract class defines the interface for general solvers that are suited for solving the [TebController](#) optimization problem.

The solver needs to handle the optimization problem as a hyper-graph (see [EdgeType](#) and [VertexType](#)).

#### Author

Christoph Rösmann ([christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de))

#### Examples:

[integrator\\_system\\_sfuns.cpp](#).

Definition at line 29 of file `base_solver.h`.

### 16.4.2 Member Typedef Documentation

#### 16.4.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer`

Definition at line 36 of file `base_solver.h`.

#### 16.4.2.2 `using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer`

Definition at line 34 of file `base_solver.h`.

### 16.4.3 Constructor & Destructor Documentation

#### 16.4.3.1 `teb::BaseSolver::BaseSolver ( ) [inline]`

Definition at line 39 of file `base_solver.h`.

16.4.3.2 `virtual teb::BaseSolver::~~BaseSolver ( ) [inline],[virtual]`

Definition at line 42 of file `base_solver.h`.

## 16.4.4 Member Function Documentation

16.4.4.1 `void teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta ) [inline],[protected]`

This method iterates the active vertices container and calls [VertexType::plusFree\(\)](#) for each vertex.

### Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.
--------------	--

Definition at line 127 of file `base_solver.h`.

References `_active_vertices`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

16.4.4.2 `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec ) [inline],[protected]`

This method iterates the active vertices container and calls [VertexType::setFree\(\)](#) for each vertex.

### Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file `base_solver.h`.

References `_active_vertices`.

16.4.4.3 `void teb::BaseSolver::backupVertices ( ) [inline],[protected]`

### See Also

[VertexType::push\(\)](#)

Definition at line 223 of file `base_solver.h`.

References `_active_vertices`, and `_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

16.4.4.4 `void teb::BaseSolver::discardBackupVertices ( ) [inline],[protected]`

### See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file `base_solver.h`.

References `_active_vertices`, and `_no_vert_backups`.

Referenced by `solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.4.4.5** `int teb::BaseSolver::getDimEqualities ( ) const` `[inline], [protected]`

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file `base_solver.h`.

References `_equalities`.

Referenced by `solve()`.

**16.4.4.6** `int teb::BaseSolver::getDimInequalities ( ) const` `[inline], [protected]`

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file `base_solver.h`.

References `_inequalities`.

Referenced by `solve()`.

**16.4.4.7** `int teb::BaseSolver::getDimObjectives ( ) const` `[inline], [protected]`

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file `base_solver.h`.

References `_objectives`.

Referenced by `solve()`.

**16.4.4.8** `Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const` `[inline], [protected]`

This method iterates the active vertices container and calls `VertexType::getDataFree()` for each vertex.

#### Remarks

Make sure `BaseSolver::_opt_vec_dim` is valid (see `solve()`).

#### Returns

`Eigen::Vector` containing all values. The length equals the dimension of all free variables (`getOptVecDimension()`).

Definition at line 162 of file `base_solver.h`.

References `_active_vertices`, and `_opt_vec_dim`.

**16.4.4.9** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline], [protected]`

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

#### See Also

`TebController::getActiveVertices()`, `getValueDimension()`

Definition at line 180 of file `base_solver.h`.

References `_active_vertices`.

Referenced by `solve()`.

**16.4.4.10** `virtual void teb::BaseSolver::initWorkspaces ( ) [inline],[protected],[virtual]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each [EdgeType](#)) or map to existing memory.

Reimplemented in [teb::BaseSolverNonlinearProgramDense](#), [teb::SolverNloptPackage](#), and [teb::BaseSolverLeastSquares](#).

Definition at line 115 of file `base_solver.h`.

Referenced by `solve()`.

**16.4.4.11** `void teb::BaseSolver::restoreVertices ( ) [inline],[protected]`

See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `_active_vertices`, and `_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.4.4.12** `void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline],[protected]`

See Also

[VertexType::top\(\)](#)

Definition at line 227 of file `base_solver.h`.

References `_active_vertices`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.4.4.13** `void teb::BaseSolver::setConfig ( const Config * config ) [inline]`

Remarks

This function is called from the TEB class within `setSolver` method.

Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file `base_solver.h`.

References `cfg`.

Referenced by `teb::TebController< p, q >::setSolver()`.

**16.4.4.14** `virtual bool teb::BaseSolver::solve ( HyperGraph * optimizable_graph ) [inline],[virtual]`

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with `solveImpl()`.

**Todo** Split `initWorkspaces` into `init` and `update` phase in order to hot-start from previous initializations.

**Parameters**

<i>optimizable_graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
--------------------------	---

Definition at line 61 of file `base_solver.h`.

References `_active_vertices`, `_equalities`, `_equalities_dim`, `_graph_structure_modified`, `_inequalities`, `_inequalities_dim`, `_no_vert_backups`, `_objective_dim`, `_objectives`, `_opt_vec_dim`, `teb::HyperGraph::activeVertices()`, `discardBackupVertices()`, `teb::HyperGraph::equalities()`, `getDimEqualities()`, `getDimInequalities()`, `getDimObjectives()`, `getOptVecDimension()`, `teb::HyperGraph::inequalities()`, `initWorkspaces()`, `teb::HyperGraph::isGraphModified()`, `teb::HyperGraph::objectives()`, and `solveImpl()`.

**16.4.4.15** `virtual bool teb::BaseSolver::solveImpl( ) [protected], [pure virtual]`

Store results to the vertices of the active vertices container.

**Return values**

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implemented in [teb::SolverSQPDense](#), [teb::SolverNloptPackage](#), [teb::SolverSQPDense](#), [teb::BaseSolverNonlinearProgramDense](#), [teb::SolverLevenbergMarquardtEigenSparse](#), [teb::BaseSolverLeastSquares](#), and [teb::SolverLevenbergMarquardtEigenDense](#).

Referenced by `solve()`.

**16.4.5 Member Data Documentation**

**16.4.5.1** `VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected]`

Definition at line 233 of file `base_solver.h`.

Referenced by `applyIncrement()`, `applyOptVec()`, `backupVertices()`, `discardBackupVertices()`, `getOptVecCopy()`, `getOptVecDimension()`, `restoreVertices()`, `restoreVerticesButKeepBackup()`, and `solve()`.

**16.4.5.2** `EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected]`

Definition at line 235 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `getDimEqualities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `solve()`.

**16.4.5.3** `int teb::BaseSolver::_equalities_dim = -1 [protected]`

Definition at line 240 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverNloptPackage::getEqConstrDimFromStorage()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.4.5.4 `bool teb::BaseSolver::_graph_structure_modified = true` [protected]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the `solve()` method.

Definition at line 250 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.4.5.5 `EdgeContainer* teb::BaseSolver::_inequalities = nullptr` [protected]

Definition at line 236 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsInEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `getDimInequalities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `solve()`.

#### 16.4.5.6 `int teb::BaseSolver::_inequalities_dim = -1` [protected]

Definition at line 241 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.4.5.7 `unsigned int teb::BaseSolver::_no_vert_backups = 0` [protected]

Definition at line 243 of file `base_solver.h`.

Referenced by `backupVertices()`, `discardBackupVertices()`, `restoreVertices()`, and `solve()`.

#### 16.4.5.8 `int teb::BaseSolver::_objective_dim = -1` [protected]

Definition at line 239 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::getValueDimension()`, and `solve()`.

#### 16.4.5.9 `EdgeContainer* teb::BaseSolver::_objectives = nullptr` [protected]

Definition at line 234 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `getDimObjectives()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::SolverNloptPackage::objectives()`, and `solve()`.



16.4.5.10 `int teb::BaseSolver::_opt_vec_dim = -1` [protected]

Definition at line 238 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `getOptVecCopy()`, `teb::SolverNloptPackage::getOptVecDimFromStorage()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

16.4.5.11 `const Config* teb::BaseSolver::cfg = nullptr`

See Also

[setConfig\(\)](#)

Definition at line 95 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `setConfig()`, and `teb::SolverSQPDense::solveImpl()`.

16.4.5.12 `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

Definition at line 253 of file `base_solver.h`.

The documentation for this class was generated from the following file:

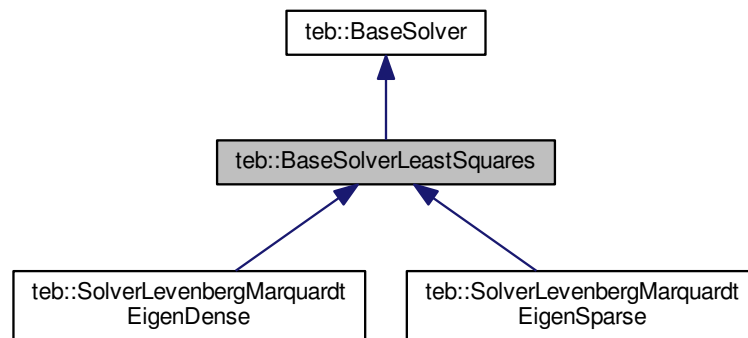
- [base\\_solver.h](#)

## 16.5 `teb::BaseSolverLeastSquares` Class Reference

Extended base solver class for least squares optimizations.

```
#include <base_solver_least_squares.h>
```

Inheritance diagram for `teb::BaseSolverLeastSquares`:



## Public Types

- using `VertexContainer` = `HyperGraph::VertexContainer`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using `EdgeContainer` = `HyperGraph::EdgeContainer`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

## Public Member Functions

- `BaseSolverLeastSquares` ()  
*Empty Constructor.*
- void `setConfig` (const `Config` \*config)  
*Set config including solver settings.*
- virtual bool `solve` (`HyperGraph` \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

## Public Attributes

- const `Config` \* `cfg` = nullptr  
*Store pointer to `Config` object.*
- `EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

## Protected Member Functions

- virtual bool `solveImpl` ()=0  
*Solve the actual optimization problem.*
- virtual void `initWorkspaces` ()  
*Initialize workspaces for the block jacobians and hessians.*
- void `buildValueVector` ()  
*Build value / cost vector including all objectives and constraints.*
- int `getValueDimension` () const  
*Get dimension of the composed value/cost vector (including objectives and constraints).*

- double [getChi2](#) () const  
*Calculate the  $\chi^2$  error of the optimization problem.*
- void [adaptWeights](#) ()  
*Automatic weight adaptation that increases soft constraint weights after each outer teb iteration.*
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const  
*Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- Eigen::VectorXd [\\_values](#)  
*Store the value vector computed in [buildValueVector\(\)](#).*
- int [\\_weight\\_adapt\\_count](#) = 0  
*Store current state of the weight adaptation method [adaptWeights\(\)](#).*
- double [\\_weight\\_equalities](#) = 1  
*Store current weight for equality soft-constraints.*
- double [\\_weight\\_inequalities](#) = 1  
*Store current weight for inequality soft-constraints.*
- int [\\_val\\_dim](#) = -1  
*Store dimension of the value vector here (see [getValueDimension\(\)](#)).*
- [VertexContainer](#) \* [\\_active\\_vertices](#) = nullptr  
*Pointer to active vertex container (active = non-fixed)*
- [EdgeContainer](#) \* [\\_objectives](#) = nullptr  
*Pointer to edges representing objective functions.*
- [EdgeContainer](#) \* [\\_equalities](#) = nullptr  
*Pointer to edges representing equality constraints.*
- [EdgeContainer](#) \* [\\_inequalities](#) = nullptr  
*Pointer to edges representing inequality constraints.*
- int [\\_opt\\_vec\\_dim](#) = -1  
*Store dimension of the optimization vector here (see [getOptVecDimension\(\)](#)).*
- int [\\_objective\\_dim](#) = -1

- *Store dimension of the objective function (see [getDimObjectives\(\)](#)).*
- `int _equalities_dim = -1`  
*Store dimension of the equality constraints (see [getDimEqualities\(\)](#)).*
- `int _inequalities_dim = -1`  
*Store dimension of the inequality constraints (see [getDimInequalities\(\)](#)).*
- `unsigned int _no_vert_backups = 0`  
*Track number of vertices backups made.*
- `bool _graph_structure_modified = true`  
*Mark if a new graph structure is available, thus we need to reinitialize all workspaces.*

### 16.5.1 Detailed Description

This abstract class extends the [BaseSolver](#) class by routines and methods required by dedicated least squares solvers.

It transforms inequality and equality constraints into soft constraints that are added to the objective functions. objective functions are squared.

The jacobian calculation is based on the new transformed objective function and used for quasi-newton hessian approximation  $\mathbf{H} = \mathbf{J}^T \mathbf{J}$ . However, the actual hessian calculation using the `computeHessian()` functions of each edge is ommitted.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

Definition at line 29 of file `base_solver_least_squares.h`.

### 16.5.2 Member Typedef Documentation

16.5.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer` [inherited]

Definition at line 36 of file `base_solver.h`.

16.5.2.2 `using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer` [inherited]

Definition at line 34 of file `base_solver.h`.

### 16.5.3 Constructor & Destructor Documentation

16.5.3.1 `teb::BaseSolverLeastSquares::BaseSolverLeastSquares ( )` [inline]

Definition at line 34 of file `base_solver_least_squares.h`.

### 16.5.4 Member Function Documentation

16.5.4.1 `void teb::BaseSolverLeastSquares::adaptWeights ( )` [protected]

This method increases the soft constraint weights for inequalities and equalities after each outer TEB optimization loop (see [TebController::optimizeTEB\(\)](#)).

After the outer TEB loop is completed ([Config::Teb::teb\\_iter](#)), the weights are set back to [Config::Optim::Solver::Lsq::weight\\_equalities](#) and [Config::Optim::Solver::Lsq::weight\\_inequalities](#).

The weights are increased by  $\sigma = \sigma_{init} \cdot \gamma^i$ .  $\gamma$  denotes the increasement factor [Config::Optim::Solver::Lsq::weight\\_adaptation\\_factor](#).  $i$  denotes the current iteration number.  $\sigma_{init}$  is obtained from the config (see above).

Call this method at the beginning of [solveImpl\(\)](#)!

#### Remarks

This procedere forces the optimization result to first contract the trajectory in sense of the objective, and afterwards trying to satisfy the constraints. Starting with very high weights in advance could lead to abrupt gradients for the solver. In addition the effect for the objectives (e.g. time optimallity) could be slow, if the corresponding gradients are extremly small in comparision to constraint gradients.

#### See Also

[TebController::optimizeTEB\(\)](#)

Definition at line 100 of file `base_solver_least_squares.cpp`.

References `_weight_adapt_count`, `_weight_equalities`, `_weight_inequalities`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::solver`, `teb::Config::teb`, `teb::Config::Teb::teb_iter`, `teb::Config::Optim::Solver::Lsq::weight_adaptation_factor`, `teb::Config::Optim::Solver::Lsq::weight_equalities`, and `teb::Config::Optim::Solver::Lsq::weight_inequalities`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.5.4.2** `void teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::plusFree\(\)](#) for each vertex.

#### Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.
--------------	--

Definition at line 127 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.5.4.3** `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::setFree\(\)](#) for each vertex.

#### Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

**16.5.4.4** `void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]`

See Also

[VertexType::push\(\)](#)

Definition at line 223 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.5.4.5 void `teb::BaseSolverLeastSquares::buildValueVector` ( ) [protected]

This method constructs the full value/cost vector  $\mathbf{f}$  (not  $\mathbf{f}^2$ ) for the underlying least-squares optimization problem.

The lenght of the vector can be obtained using [getValueDimension\(\)](#).

Constraints are transformed into costs/objectives using soft constraints:

- Equality constraints  $c(x) = 0$ .

These constraints are directly taken as objectives since  $c^2(x)$  has a local minima at  $x = 0$ .

- Inequality constraints  $c(x) \leq 0$

These constraints are transformed using  $f = \max(c(x), 0)$  (component-wise).

This notation is similar to: `c(x) <= epsilon ? 0 : c(x)`.

Afterwards (in the actual [solveImpl\(\)](#) method, the cost function will be squared, that leads to twice differnetiable costs for the constraints, if  $f$  is piecewise differentiable once and if it intersects with the abscissa.

In addition the weights [BaseSolverLeastSquares::\\_weight\\_equalities](#) and [BaseSolverLeastSquares::\\_weight\\_inequalities](#) are taken into account in order to weight the "soft constraint objectives". To make the weights comparable to [2] and our Matlab TEB version, we take the square root of both weights, since the least-square problem here is formulated as  $f^2(x, \sigma)$  instead of  $\sigma f^2(x)$ .  $\sigma$  denotes the weight.

The results are stored internally to [BaseSolverLeastSquares::\\_values](#).

See Also

[adaptWeights\(\)](#), [getValueDimension\(\)](#)

Definition at line 31 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `_val_dim`, `_values`, `_weight_equalities`, `_weight_inequalities`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::Solver::Lsq::soft_constr_epsilon`, and `teb::Config::Optim::solver`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

#### 16.5.4.6 void `teb::BaseSolver::discardBackupVertices` ( ) [inline],[protected],[inherited]

See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.5.4.7** `double teb::BaseSolverLeastSquares::getChi2( ) const [inline],[protected]`

The  $\chi^2$  error is the scalar cost value of the least-squares optimization problem since the total cost function is composed of weighted terms:

$$f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \mathbf{f}(\mathcal{B})$$

Weights are included in  $\mathbf{f}(\mathcal{B})$ . In this case it is  $\chi^2 = f(\mathcal{B})$ .

**Todo** Maybe switch to the formulation  $f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \Omega \mathbf{f}(\mathcal{B})$  similar to g2o and Teb-Matlab.

Definition at line 82 of file `base_solver_least_squares.h`.

References `_values`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.5.4.8** `int teb::BaseSolver::getDimEqualities( ) const [inline],[protected],[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file `base_solver.h`.

References `teb::BaseSolver::_equalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.5.4.9** `int teb::BaseSolver::getDimInequalities( ) const [inline],[protected],[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file `base_solver.h`.

References `teb::BaseSolver::_inequalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.5.4.10** `int teb::BaseSolver::getDimObjectives( ) const [inline],[protected],[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file `base_solver.h`.

References `teb::BaseSolver::_objectives`.

Referenced by `teb::BaseSolver::solve()`.

**16.5.4.11** `Eigen::VectorXd teb::BaseSolver::getOptVecCopy( ) const [inline],[protected],[inherited]`

This method iterates the active vertices container and calls `VertexType::getDataFree()` for each vertex.

#### Remarks

Make sure `BaseSolver::_opt_vec_dim` is valid (see `solve()`).

#### Returns

`Eigen::VectorXd` containing all values. The length equals the dimension of all free variables (`getOptVecDimension()`).

Definition at line 162 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_opt_vec_dim`.

**16.5.4.12** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking the previously determined index of the last active vertex stored in the graph.

See Also

[TebController::getActiveVertices\(\)](#), [getValueDimension\(\)](#)

Definition at line 180 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

**16.5.4.13** `int teb::BaseSolverLeastSquares::getValueDimension ( ) const` `[protected]`

This method returns the dimension of the full cost/value vector including all objective and constraints.

In particular it sums up all dimensions of single edges stored in the given hyper-graph.

Remarks

Make sure [BaseSolver::\\_objective\\_dim](#), [BaseSolver::\\_equalities\\_dim](#) and [BaseSolver::\\_inequalities\\_dim](#) are valid (see [solve\(\)](#)).

Returns

dimension of the complete value/cost vector.

See Also

[buildValueVector\(\)](#), [getOptVecDimension\(\)](#), [getDimObjectives\(\)](#), [getDimEqualities\(\)](#), [getDimInequalities\(\)](#)

Definition at line 73 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_inequalities_dim`, and `teb::BaseSolver::_objective_dim`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.5.4.14** `virtual void teb::BaseSolverLeastSquares::initWorkspaces ( )` `[inline]`, `[protected]`, `[virtual]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each [Edge-Type](#)) or map to existing memory.

Reimplemented from [teb::BaseSolver](#).

Definition at line 43 of file `base_solver_least_squares.h`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `PRINT_DEBUG_COND_ONCE`, and `teb::QUADRATIC`.

**16.5.4.15** `void teb::BaseSolver::restoreVertices ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.



16.5.4.16 void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline],[protected],[inherited]

See Also

[VertexType::top\(\)](#)

Definition at line 227 of file base\_solver.h.

References [teb::BaseSolver::\\_active\\_vertices](#).

Referenced by [teb::SolverSQPDense::solveImpl\(\)](#).

16.5.4.17 void teb::BaseSolver::setConfig ( const Config \* config ) [inline],[inherited]

Remarks

This function is called from the TEB class within setSolver method.

Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file base\_solver.h.

References [teb::BaseSolver::cfg](#).

Referenced by [teb::TebController< p, q >::setSolver\(\)](#).

16.5.4.18 virtual bool teb::BaseSolver::solve ( HyperGraph \* optimizable\_graph ) [inline],[virtual],[inherited]

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with [solveImpl\(\)](#).

**Todo** Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

Parameters

<i>optimizable_graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
--------------------------	---

Definition at line 61 of file base\_solver.h.

References [teb::BaseSolver::\\_active\\_vertices](#), [teb::BaseSolver::\\_equalities](#), [teb::BaseSolver::\\_equalities\\_dim](#), [teb::BaseSolver::\\_graph\\_structure\\_modified](#), [teb::BaseSolver::\\_inequalities](#), [teb::BaseSolver::\\_inequalities\\_dim](#), [teb::BaseSolver::\\_no\\_vert\\_backups](#), [teb::BaseSolver::\\_objective\\_dim](#), [teb::BaseSolver::\\_objectives](#), [teb::BaseSolver::\\_opt\\_vec\\_dim](#), [teb::HyperGraph::activeVertices\(\)](#), [teb::BaseSolver::discardBackupVertices\(\)](#), [teb::HyperGraph::equalities\(\)](#), [teb::BaseSolver::getDimEqualities\(\)](#), [teb::BaseSolver::getDimInequalities\(\)](#), [teb::BaseSolver::getDimObjectives\(\)](#), [teb::BaseSolver::getOptVecDimension\(\)](#), [teb::HyperGraph::inequalities\(\)](#), [teb::BaseSolver::initWorkspaces\(\)](#), [teb::HyperGraph::isGraphModified\(\)](#), [teb::HyperGraph::objectives\(\)](#), and [teb::BaseSolver::solveImpl\(\)](#).

16.5.4.19 virtual bool teb::BaseSolverLeastSquares::solveImpl ( ) [protected],[pure virtual]

Store results to the vertices of the active vertices container.

## Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolver](#).

Implemented in [teb::SolverLevenbergMarquardtEigenSparse](#), and [teb::SolverLevenbergMarquardtEigenDense](#).

### 16.5.5 Member Data Documentation

#### 16.5.5.1 `VertexContainer* teb::BaseSolver::_active_vertices = nullptr` [protected],[inherited]

Definition at line 233 of file `base_solver.h`.

Referenced by `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::applyOptVec()`, `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::getOptVecCopy()`, `teb::BaseSolver::getOptVecDimension()`, `teb::BaseSolver::restoreVertices()`, `teb::BaseSolver::restoreVerticesButKeepBackup()`, and `teb::BaseSolver::solve()`.

#### 16.5.5.2 `EdgeContainer* teb::BaseSolver::_equalities = nullptr` [protected],[inherited]

Definition at line 235 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimEqualities()`, `initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

#### 16.5.5.3 `int teb::BaseSolver::_equalities_dim = -1` [protected],[inherited]

Definition at line 240 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverNloptPackage::getEqConstrDimFromStorage()`, `getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.5.5.4 `bool teb::BaseSolver::_graph_structure_modified = true` [protected],[inherited]

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the `solve()` method.

Definition at line 250 of file `base_solver.h`.

Referenced by `initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.5.5.5 `EdgeContainer* teb::BaseSolver::_inequalities = nullptr` [protected],[inherited]

Definition at line 236 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraints-InEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimInequalities()`, `initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

#### 16.5.5.6 `int teb::BaseSolver::_inequalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 241 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.5.5.7 `unsigned int teb::BaseSolver::_no_vert_backups = 0` `[protected]`, `[inherited]`

Definition at line 243 of file `base_solver.h`.

Referenced by `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::restoreVertices()`, and `teb::BaseSolver::solve()`.

#### 16.5.5.8 `int teb::BaseSolver::_objective_dim = -1` `[protected]`, `[inherited]`

Definition at line 239 of file `base_solver.h`.

Referenced by `getValueDimension()`, and `teb::BaseSolver::solve()`.

#### 16.5.5.9 `EdgeContainer* teb::BaseSolver::_objectives = nullptr` `[protected]`, `[inherited]`

Definition at line 234 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimObjectives()`, `initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::SolverNloptPackage::objectives()`, and `teb::BaseSolver::solve()`.

#### 16.5.5.10 `int teb::BaseSolver::_opt_vec_dim = -1` `[protected]`, `[inherited]`

Definition at line 238 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getOptVecCopy()`, `teb::SolverNloptPackage::getOptVecDimFromStorage()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.5.5.11** `int teb::BaseSolverLeastSquares::_val_dim = -1` `[protected]`

Definition at line 96 of file `base_solver_least_squares.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.5.5.12** `Eigen::VectorXd teb::BaseSolverLeastSquares::_values` `[protected]`

Definition at line 90 of file `base_solver_least_squares.h`.

Referenced by `buildValueVector()`, `getChi2()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.5.5.13** `int teb::BaseSolverLeastSquares::_weight_adapt_count = 0` `[protected]`

Definition at line 92 of file `base_solver_least_squares.h`.

Referenced by `adaptWeights()`.

**16.5.5.14** `double teb::BaseSolverLeastSquares::_weight_equalities = 1` `[protected]`

Definition at line 93 of file `base_solver_least_squares.h`.

Referenced by `adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, and `buildValueVector()`.

**16.5.5.15** `double teb::BaseSolverLeastSquares::_weight_inequalities = 1` `[protected]`

Definition at line 94 of file `base_solver_least_squares.h`.

Referenced by `adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, and `buildValueVector()`.

**16.5.5.16** `const Config* teb::BaseSolver::cfg = nullptr` `[inherited]`

See Also

[setConfig\(\)](#)

Definition at line 95 of file `base_solver.h`.

Referenced by `adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::setConfig()`, and `teb::SolverSQPDense::solveImpl()`.

**16.5.5.17** `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW` `[inherited]`

Definition at line 253 of file `base_solver.h`.

The documentation for this class was generated from the following files:

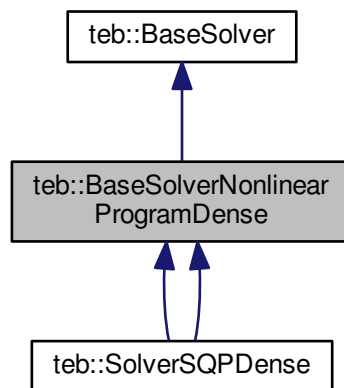
- [base\\_solver\\_least\\_squares.h](#)
- [base\\_solver\\_least\\_squares.cpp](#)

## 16.6 teb::BaseSolverNonlinearProgramDense Class Reference

Extended base solver class for nonlinear programs (dense version).

```
#include <base_solver_nonlinear_program_dense.h>
```

Inheritance diagram for teb::BaseSolverNonlinearProgramDense:



### Public Types

- using [VertexContainer](#) = [HyperGraph::VertexContainer](#)  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using [EdgeContainer](#) = [HyperGraph::EdgeContainer](#)  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

### Public Member Functions

- [BaseSolverNonlinearProgramDense](#) ()  
*Empty Constructor.*
- void [setConfig](#) (const [Config](#) \*config)  
*Set config including solver settings.*
- virtual bool [solve](#) ([HyperGraph](#) \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

### Public Attributes

- const [Config](#) \* [cfg](#) = nullptr  
*Store pointer to Config object.*
- [EIGEN\\_MAKE\\_ALIGNED\\_OPERATOR\\_NEW](#)

### Protected Types

- using [MatMapRowMajor](#) = [Eigen::Map](#)< [Eigen::Matrix](#)< double,-1,-1, [Eigen::RowMajor](#) >>

## Protected Member Functions

- virtual bool [solveImpl](#) ()=0  
*Solve the actual optimization problem.*
- virtual void [initWorkspaces](#) ()  
*Initialize workspaces for the block jacobians and hessians.*
- virtual void [initSolverWorkspace](#) ()=0
- void [initializeLagrangeMultiplier](#) ()
- void [buildObjectiveValue](#) ()  
*Get the sum of all objective values since we solve  $f = \min \sum f_k$ .*
- void [buildEqualityConstraintValueVector](#) ()  
*Build value / cost vector including all equality constraints.*
- void [buildInequalityConstraintValueVector](#) ()  
*Build value / cost vector including all inequality constraints.*
- void [buildObjectiveGradient](#) ()
- void [buildEqualityConstraintJacobian](#) ()
- void [buildInequalityConstraintJacobian](#) ()
- void [calculateLagrangianGradient](#) ()
- void [calculateLagrangianHessian](#) (Eigen::VectorXd \*increment=nullptr)
- void [calculateLagrangianHessianNumerically](#) ()
- void [initHessianBFGS](#) ()
- void [calculateLagrangianHessianFullBFGS](#) (Eigen::VectorXd \*increment)
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const  
*Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- double `_objective_value`  
Store the sum of all values computed in `buildObjectiveValueVector()` since we solve  $f = \min \sum f_k$ .
- Eigen::VectorXd `_equality_values`  
Store the value vector computed in `buildEqualityConstraintValueVector()`.
- Eigen::VectorXd `_inequality_values`  
Store the value vector computed in `buildInequalityConstraintValueVector()`.
- Eigen::VectorXd `_objective_gradient`  
Store the gradient of the objective value computed in `buildObjectiveGradient()`.
- MatMapRowMajor `_equality_jacobian` = MatMapRowMajor(nullptr,0,0)  
Store the jacobian matrix of the equality constraint computed in `buildEqualityConstraintJacobian()`.
- MatMapRowMajor `_inequality_jacobian` = MatMapRowMajor(nullptr,0,0)  
Store the jacobian matrix of the inequality constraint computed in `buildInequalityConstraintJacobian()`.
- Eigen::VectorXd `_lagrangian_gradient`
- Eigen::VectorXd `_lagrangian_gradient_backup`  
The gradient from the last step has to be stored for BFGS.
- Eigen::Matrix< double,-1,-1,  
Eigen::RowMajor > `_lagrangian_hessian`  
Store the hessian of the lagrangian  $\nabla^2 L = \nabla^2 (f - \mu^T \mathbf{ceq} - \lambda^T \mathbf{c})$ .
- Eigen::VectorXd `_multiplier_ineq`
- Eigen::VectorXd `_multiplier_eq`
- VertexContainer \* `_active_vertices` = nullptr  
Pointer to active vertex container (active = non-fixed)
- EdgeContainer \* `_objectives` = nullptr  
Pointer to edges representing objective functions.
- EdgeContainer \* `_equalities` = nullptr  
Pointer to edges representing equality constraints.
- EdgeContainer \* `_inequalities` = nullptr  
Pointer to edges representing inequality constraints.
- int `_opt_vec_dim` = -1  
Store dimension of the optimization vector here (see `getOptVecDimension()`).
- int `_objective_dim` = -1  
Store dimension of the objective function (see `getDimObjectives()`).
- int `_equalities_dim` = -1  
Store dimension of the equality constraints (see `getDimEqualities()`).
- int `_inequalities_dim` = -1  
Store dimension of the inequality constraints (see `getDimInequalities()`).
- unsigned int `_no_vert_backups` = 0  
Track number of vertices backups made.
- bool `_graph_structure_modified` = true  
Mark if a new graph structure is available, thus we need to reinitialize all workspaces.

### 16.6.1 Detailed Description

This abstract class extends the [BaseSolver](#) class by routines and methods required by dedicated nonlinear program solvers.

Inequality and equality constraints are treated as hard constraints. This class provides methods to calculate objective and constraint gradients/jacobians and as well as the Hessian of the Lagrangian.

The actual `solveImpl()` implementation has to be done in a separate subclass, e.g. depending if the underlying nonlinear-program solver is a newton-type solver, interior-point, barrier, sqp, ... solver.

**Todo** The documentation of this class, after this class passed a couple of more tests

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

Definition at line 32 of file base\_solver\_nonlinear\_program\_dense.h.

### 16.6.2 Member Typedef Documentation

16.6.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer` [inherited]

Definition at line 36 of file base\_solver.h.

16.6.2.2 `using teb::BaseSolverNonlinearProgramDense::MatMapRowMajor = Eigen::Map<Eigen::Matrix<double,-1,-1,Eigen::RowMajor>>` [protected]

Definition at line 110 of file base\_solver\_nonlinear\_program\_dense.h.

16.6.2.3 `using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer` [inherited]

Definition at line 34 of file base\_solver.h.

### 16.6.3 Constructor & Destructor Documentation

16.6.3.1 `teb::BaseSolverNonlinearProgramDense::BaseSolverNonlinearProgramDense ( )` [inline]

Definition at line 37 of file base\_solver\_nonlinear\_program\_dense.h.

### 16.6.4 Member Function Documentation

16.6.4.1 `void teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta )` [inline], [protected], [inherited]

This method iterates the active vertices container and calls `VertexType::plusFree()` for each vertex.

#### Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length equals the dimension of all free variables.
--------------	---

Definition at line 127 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

16.6.4.2 `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec )` [inline], [protected], [inherited]

This method iterates the active vertices container and calls `VertexType::setFree()` for each vertex.



## Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`.

**16.6.4.3** `void teb::BaseSolver::backupVertices ( )` `[inline]`, `[protected]`, `[inherited]`

## See Also

[VertexType::push\(\)](#)

Definition at line 223 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.4.4** `void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian ( )` `[protected]`

Definition at line 154 of file base\_solver\_nonlinear\_program\_dense.cpp.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `_equality_jacobian`, `_equality_values`, `teb::BaseSolver::_opt_vec_dim`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, and `teb::VertexType::isFixedComp()`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.5** `void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector ( )` `[protected]`

Definition at line 30 of file base\_solver\_nonlinear\_program\_dense.cpp.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, and `_equality_values`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.6** `void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian ( )` `[protected]`

Definition at line 200 of file base\_solver\_nonlinear\_program\_dense.cpp.

References `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, `_inequality_jacobian`, `_inequality_values`, `teb::BaseSolver::_opt_vec_dim`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, and `teb::VertexType::isFixedComp()`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.7** `void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector ( )` `[protected]`

Definition at line 54 of file base\_solver\_nonlinear\_program\_dense.cpp.

References `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, and `_inequality_values`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.8** `void teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient ( )` `[protected]`

Definition at line 82 of file base\_solver\_nonlinear\_program\_dense.cpp.

References `_objective_gradient`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, `teb::VertexType::isFixedComp()`, `teb::LINEAR_SQUARED`, and `teb::NONLINEAR_SQUARED`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.9** `void teb::BaseSolverNonlinearProgramDense::buildObjectiveValue ( ) [protected]`

Definition at line 7 of file `base_solver_nonlinear_program_dense.cpp`.

References `_objective_value`, `teb::BaseSolver::_objectives`, `teb::LINEAR_SQUARED`, and `teb::NONLINEAR_SQUARED`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.10** `void teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient ( ) [inline], [protected]`

Definition at line 98 of file `base_solver_nonlinear_program_dense.h`.

References `_equality_jacobian`, `_inequality_jacobian`, `_lagrangian_gradient`, `_multiplier_eq`, `_multiplier_ineq`, and `_objective_gradient`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.11** `void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian ( Eigen::VectorXd * increment = nullptr ) [protected]`

Definition at line 245 of file `base_solver_nonlinear_program_dense.cpp`.

References `teb::BLOCK_BFGS`, `calculateLagrangianHessianFullBFGS()`, `calculateLagrangianHessianNumerically()`, `teb::BaseSolver::cfg`, `teb::FULL_BFGS`, `teb::FULL_BFGS_WITH_STRUCTURE_FILTER`, `teb::Config::Optim::Solver::NonlinearProgram::hessian`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method`, `teb::Config::Optim::Solver::nonlin_prog`, `teb::NUMERIC`, `teb::Config::optim`, `PRINT_ERROR`, `PRINT_INFO`, `teb::Config::Optim::solver`, and `teb::ZERO_HESSIAN`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.12** `void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS ( Eigen::VectorXd * increment ) [protected]`

Definition at line 520 of file `base_solver_nonlinear_program_dense.cpp`.

References `_lagrangian_gradient`, `_lagrangian_gradient_backup`, `_lagrangian_hessian`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::bfgs_damped_mode`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::NonlinearProgram::hessian`, `teb::Config::Optim::Solver::nonlin_prog`, `teb::Config::optim`, `PRINT_DEBUG`, and `teb::Config::Optim::solver`.

Referenced by `calculateLagrangianHessian()`.

**16.6.4.13** `void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically ( ) [protected]`

Definition at line 273 of file `base_solver_nonlinear_program_dense.cpp`.

References `teb::BaseSolver::equalities`, `teb::BaseSolver::equalities_dim`, `_equality_values`, `teb::BaseSolver::inequalities`, `teb::BaseSolver::inequalities_dim`, `_inequality_values`, `_lagrangian_hessian`, `_multiplier_eq`, `_multiplier_ineq`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, `teb::VertexType::isFixedComp()`, `teb::LINEAR`, `teb::LINEAR_SQUARED`, `teb::NONLINEAR_ONCE_DIFF`, and `teb::NONLINEAR_SQUARED`.

Referenced by `calculateLagrangianHessian()`, and `initHessianBFGS()`.

**16.6.4.14** `void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]`

See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.4.15** `int teb::BaseSolver::getDimEqualities ( ) const [inline], [protected], [inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file `base_solver.h`.

References `teb::BaseSolver::_equalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.6.4.16** `int teb::BaseSolver::getDimInequalities ( ) const [inline], [protected], [inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file `base_solver.h`.

References `teb::BaseSolver::_inequalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.6.4.17** `int teb::BaseSolver::getDimObjectives ( ) const [inline], [protected], [inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file `base_solver.h`.

References `teb::BaseSolver::_objectives`.

Referenced by `teb::BaseSolver::solve()`.

**16.6.4.18** `Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::getDataFree\(\)](#) for each vertex.

Remarks

Make sure [BaseSolver::\\_opt\\_vec\\_dim](#) is valid (see [solve\(\)](#)).

Returns

`Eigen::Vector` containing all values. The length equals the dimension of all free variables ([getOptVecDimension\(\)](#)).

Definition at line 162 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_opt_vec_dim`.

**16.6.4.19** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline], [protected], [inherited]`

The dimension is obtained by checking the previously determined index of the last active vertex stored in the graph.

See Also

[TebController::getActiveVertices\(\)](#), [getValueDimension\(\)](#)

Definition at line 180 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

**16.6.4.20** `void teb::BaseSolverNonlinearProgramDense::initHessianBFGS ( )` `[protected]`

Definition at line 489 of file `base_solver_nonlinear_program_dense.cpp`.

References `_lagrangian_gradient`, `_lagrangian_gradient_backup`, `_lagrangian_hessian`, `teb::BaseSolver::_opt_vec_dim`, `calculateLagrangianHessianNumerically()`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::NonlinearProgram::hessian`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init_identity_scale`, `teb::IDENTITY`, `teb::Config::Optim::Solver::nonlin_prog`, `teb::NUMERIC`, `teb::Config::optim`, `PRINT_ERROR`, `teb::Config::Optim::solver`, and `teb::ZERO`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.21** `void teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier ( )` `[inline], [protected]`

Definition at line 70 of file `base_solver_nonlinear_program_dense.h`.

References `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_inequalities_dim`, `_multiplier_eq`, and `_multiplier_ineq`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.22** `virtual void teb::BaseSolverNonlinearProgramDense::initSolverWorkspace ( )` `[protected], [pure virtual]`

Implemented in [teb::SolverSQPDense](#), and [teb::SolverSQPDense](#).

**16.6.4.23** `virtual void teb::BaseSolverNonlinearProgramDense::initWorkspaces ( )` `[inline], [protected], [virtual]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each [Edge-Type](#)) or map to existing memory.

Reimplemented from [teb::BaseSolver](#).

Definition at line 46 of file `base_solver_nonlinear_program_dense.h`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `teb::LINEAR_SQUARED`, `teb::NONLINEAR_SQUARED`, and `PRINT_DEBUG_COND_ONCE`.

**16.6.4.24** `void teb::BaseSolver::restoreVertices ( )` `[inline], [protected], [inherited]`

See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.6.4.25** `void teb::BaseSolver::restoreVerticesButKeepBackup ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::top\(\)](#)

Definition at line 227 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.6.4.26** `void teb::BaseSolver::setConfig ( const Config * config )` `[inline]`, `[inherited]`

Remarks

This function is called from the TEB class within `setSolver` method.

Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file `base_solver.h`.

References `teb::BaseSolver::cfg`.

Referenced by `teb::TebController< p, q >::setSolver()`.

**16.6.4.27** `virtual bool teb::BaseSolver::solve ( HyperGraph * optimizable_graph )` `[inline]`, `[virtual]`, `[inherited]`

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with [solveImpl\(\)](#).

**Todo** Split `initWorkspaces` into `init` and `update` phase in order to hot-start from previous initializations.

Parameters

<i>optimizable_ - graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
-----------------------------	---

Definition at line 61 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolver::_no_vert_backups`, `teb::BaseSolver::_objective_dim`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::HyperGraph::activeVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::HyperGraph::equalities()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolver::getOptVecDimension()`, `teb::HyperGraph::inequalities()`, `teb::BaseSolver::initWorkspaces()`, `teb::HyperGraph::isGraphModified()`, `teb::HyperGraph::objectives()`, and `teb::BaseSolver::solveImpl()`.

16.6.4.28 `virtual bool teb::BaseSolverNonlinearProgramDense::solveImpl ( )` [protected],[pure virtual]

Store results to the vertices of the active vertices container.

## Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolver](#).

Implemented in [teb::SolverSQPDense](#), and [teb::SolverSQPDense](#).

### 16.6.5 Member Data Documentation

#### 16.6.5.1 `VertexContainer*` `teb::BaseSolver::_active_vertices = nullptr` `[protected]`, `[inherited]`

Definition at line 233 of file `base_solver.h`.

Referenced by `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::applyOptVec()`, `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::getOptVecCopy()`, `teb::BaseSolver::getOptVecDimension()`, `teb::BaseSolver::restoreVertices()`, `teb::BaseSolver::restoreVerticesButKeepBackup()`, and `teb::BaseSolver::solve()`.

#### 16.6.5.2 `EdgeContainer*` `teb::BaseSolver::_equalities = nullptr` `[protected]`, `[inherited]`

Definition at line 235 of file `base_solver.h`.

Referenced by `buildEqualityConstraintJacobian()`, `buildEqualityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `initWorkspaces()`, and `teb::BaseSolver::solve()`.

#### 16.6.5.3 `int` `teb::BaseSolver::_equalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 240 of file `base_solver.h`.

Referenced by `buildEqualityConstraintJacobian()`, `buildEqualityConstraintValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverNloptPackage::getEqConstrDimFromStorage()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.6.5.4 `MatMapRowMajor` `teb::BaseSolverNonlinearProgramDense::_equality_jacobian = MatMapRowMajor(nullptr,0,0)` `[protected]`

Definition at line 117 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `buildEqualityConstraintJacobian()`, `calculateLagrangianGradient()`, `teb::SolverSQPDense::calculateMeritDerivative()`, and `teb::SolverSQPDense::initSolverWorkspace()`.

#### 16.6.5.5 `Eigen::VectorXd` `teb::BaseSolverNonlinearProgramDense::_equality_values` `[protected]`

Definition at line 113 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `buildEqualityConstraintJacobian()`, `buildEqualityConstraintValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMerit()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverSQPDense::checkConvergence()`, `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.6** `bool teb::BaseSolver::_graph_structure_modified = true` `[protected]`, `[inherited]`

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the `solve()` method.

Definition at line 250 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::initWorkspaces()`, `initWorkspaces()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.7** `EdgeContainer* teb::BaseSolver::_inequalities = nullptr` `[protected]`, `[inherited]`

Definition at line 236 of file `base_solver.h`.

Referenced by `buildInequalityConstraintJacobian()`, `buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsInEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `initWorkspaces()`, and `teb::BaseSolver::solve()`.

**16.6.5.8** `int teb::BaseSolver::_inequalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 241 of file `base_solver.h`.

Referenced by `buildInequalityConstraintJacobian()`, `buildInequalityConstraintValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.9** `MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_inequality_jacobian = MatMapRowMajor(nullptr,0,0)` `[protected]`

Definition at line 118 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `buildInequalityConstraintJacobian()`, `calculateLagrangianGradient()`, `teb::SolverSQPDense::calculateMeritDerivative()`, and `teb::SolverSQPDense::initSolverWorkspace()`.

**16.6.5.10** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_inequality_values` `[protected]`

Definition at line 114 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `buildInequalityConstraintJacobian()`, `buildInequalityConstraintValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMerit()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.11** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient` `[protected]`

Definition at line 120 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `calculateLagrangianGradient()`, `calculateLagrangianHessianFullBFGS()`, `teb::SolverSQPDense::checkConvergence()`, and `initHessianBFGS()`.

**16.6.5.12** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup` `[protected]`

Definition at line 121 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `calculateLagrangianHessianFullBFGS()`, and `initHessianBFGS()`.



**16.6.5.13** `Eigen::Matrix<double,-1,-1,Eigen::RowMajor> teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian` `[protected]`

Definition at line 122 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `calculateLagrangianHessianFullBFGS()`, `calculateLagrangianHessianNumerically()`, `initHessianBFGS()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.14** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_eq` `[protected]`

Definition at line 125 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `calculateLagrangianGradient()`, `calculateLagrangianHessianNumerically()`, `initializeLagrangeMultiplier()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.15** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_ineq` `[protected]`

Definition at line 124 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `calculateLagrangianGradient()`, `calculateLagrangianHessianNumerically()`, `initializeLagrangeMultiplier()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.16** `unsigned int teb::BaseSolver::_no_vert_backups = 0` `[protected]`, `[inherited]`

Definition at line 243 of file `base_solver.h`.

Referenced by `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::restoreVertices()`, and `teb::BaseSolver::solve()`.

**16.6.5.17** `int teb::BaseSolver::_objective_dim = -1` `[protected]`, `[inherited]`

Definition at line 239 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::solve()`.

**16.6.5.18** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_objective_gradient` `[protected]`

Definition at line 116 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `buildObjectiveGradient()`, `calculateLagrangianGradient()`, `teb::SolverSQPDense::calculateMerit()`, `teb::SolverSQPDense::calculateMeritDerivative()`, and `teb::SolverSQPDense::solveImpl()`.

**16.6.5.19** `double teb::BaseSolverNonlinearProgramDense::_objective_value` `[protected]`

Definition at line 112 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `buildObjectiveValue()`, and `teb::SolverSQPDense::calculateMerit()`.

**16.6.5.20** `EdgeContainer* teb::BaseSolver::_objectives = nullptr` `[protected]`, `[inherited]`

Definition at line 234 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `buildObjectiveGradient()`, `buildObjectiveValue()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolverLeastSquares::initWorkspaces()`,

teb::SolverNloptPackage::initWorkspaces(), initWorkspaces(), teb::SolverNloptPackage::objectives(), and teb::BaseSolver::solve().

**16.6.5.21** `int teb::BaseSolver::_opt_vec_dim = -1` [protected],[inherited]

Definition at line 238 of file base\_solver.h.

Referenced by teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian(), buildEqualityConstraintJacobian(), buildInequalityConstraintJacobian(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), buildObjectiveGradient(), calculateLagrangianHessianNumerically(), teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ(), teb::BaseSolver::getOptVecCopy(), teb::SolverNloptPackage::getOptVecDimFromStorage(), initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::solve(), teb::SolverLevenbergMarquardtEigenDense::solveImpl(), teb::SolverLevenbergMarquardtEigenSparse::solveImpl(), and teb::SolverSQPDense::solveImpl().

**16.6.5.22** `const Config* teb::BaseSolver::cfg = nullptr` [inherited]

See Also

[setConfig\(\)](#)

Definition at line 95 of file base\_solver.h.

Referenced by teb::BaseSolverLeastSquares::adaptWeights(), teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), teb::BaseSolverLeastSquares::buildValueVector(), calculateLagrangianHessian(), calculateLagrangianHessianFullBFGS(), initHessianBFGS(), teb::SolverSQPDense::initSolverWorkspace(), teb::BaseSolver::setConfig(), and teb::SolverSQPDense::solveImpl().

**16.6.5.23** `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW` [inherited]

Definition at line 253 of file base\_solver.h.

The documentation for this class was generated from the following files:

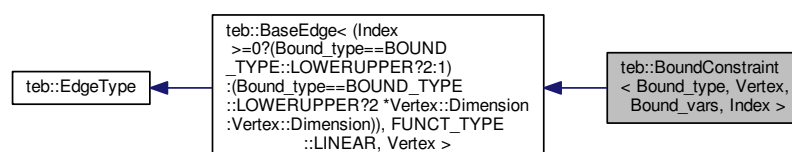
- [base\\_solver\\_nonlinear\\_program\\_dense.h](#)
- [base\\_solver\\_nonlinear\\_program\\_dense.cpp](#)

## 16.7 teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index > Class Template Reference

This class captures general bound constraints on optimization variables / vertices.

```
#include <bound_constraints.h>
```

Inheritance diagram for teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index >:



## Public Types

- using [ValueVector](#) = typename [BaseEdge](#)< [NoBounds](#), [FUNCT\\_TYPE::LINEAR](#), Vertex >::ValueVector  
*Typedef to represent the static Eigen::Vector of cost/constraint values.*
- using [EdgeContainer](#) = std::vector< [EdgeType](#) \* >  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*
- using [ValueVectorMap](#) = Eigen::Map< Eigen::VectorXd >  
*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

## Public Member Functions

- [BoundConstraint](#) (Vertex &vertex)  
*Construct edge and attach the corresponding vertex.*
- virtual const bool [isBoundConstraint](#) () const  
*Override this method to specify this edge as a bound constraint since it can be handled separately by some solvers to speed up optimization.*
- virtual void [computeValues](#) ()  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual void [computeJacobian](#) ()  
*Specification of the analytic block jacobian.*
- virtual void [computeHessian](#) ()  
*Specification of the analytic block hessian.*
- void [setBounds](#) (const double \*bounds)  
*Set bounds for either [BOUND\\_TYPE::LOWER](#) or [BOUND\\_TYPE::UPPER](#).*
- void [setBounds](#) (const Eigen::Ref< const Eigen::Matrix< double, [NoBounds](#), 1 >> &bounds)  
*Set bounds for either [BOUND\\_TYPE::LOWER](#) or [BOUND\\_TYPE::UPPER](#).*
- void [setBounds](#) (double bound)  
*Set single bound for either [BOUND\\_TYPE::LOWER](#) or [BOUND\\_TYPE::UPPER](#).*
- void [setBounds](#) (double lb, double ub)  
*Set single bound for [BOUND\\_TYPE::LOWERUPPER](#).*
- void [setBounds](#) (const double \*lb, const double \*ub)  
*Set lower and upper bounds for [BOUND\\_TYPE::LOWERUPPER](#).*
- void [setBounds](#) (const Eigen::Ref< const Eigen::Matrix< double, [NoBounds](#)/2, 1 >> &lb, const Eigen::Ref< const Eigen::Matrix< double, [NoBounds](#)/2, 1 >> &ub)  
*Set lower and upper bounds for [BOUND\\_TYPE::LOWERUPPER](#).*
- const double \* [getBounds](#) () const  
*get a constant pointer to bound data.*
- virtual void \* [getCustomData](#) ()  
*Return bound information.*
- virtual int [dimension](#) () const  
*Return edge dimension by function call.*
- virtual [FUNCT\\_TYPE](#) [funcType](#) () const  
*Return function typed given by template parameter FuncType.*
- virtual const double \* [valuesData](#) () const  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1] (read-only).*
- virtual double \* [valuesData](#) ()  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1].*
- virtual const [ValueVector](#) & [values](#) () const  
*Return cost/constraint values as static Eigen::Vector (read-only).*
- virtual [ValueVector](#) & [values](#) ()  
*Return cost/constraint values as static Eigen::Vector.*

- virtual void `allocateMemory` (bool skip\_hessian=false)  
*Allocate memory for `JacobianWorkspace` and `HessianWorkspace` and get vertices dimensions.*
- virtual `VertexType` \* `getVertex` (unsigned int idx)  
*Access attached vertex with index `idx`.*
- virtual const `VertexType` \* `getVertex` (unsigned int idx) const  
*Access attached vertex with index `idx` (read-only).*
- virtual unsigned int `noVertices` () const  
*Return the number of attached vertices.*
- virtual void `calculateVertexDimensions` ()  
*Return the number of attached vertices.*
- virtual const std::array  
< `VertexType` \*, `NoVertices` > & `vertices` () const  
*Return vertex container of the edge.*
- virtual `ValueVectorMap` `valuesMap` ()  
*Get cost/constraints values as Eigen type matrix.*
- virtual `JacobianWorkspace` & `jacobians` ()  
*Access the jacobian workspace of this edge.*
- virtual const `JacobianWorkspace` & `jacobians` () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual `HessianWorkspace` & `hessians` ()  
*Access the hessian workspace of this edge.*
- virtual const `HessianWorkspace` & `hessians` () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual void `backupJacobian` ()  
*Make a backup of the current jacobian block matrix.*
- virtual void `restoreJacobian` ()  
*Restore jacobian block matrix from the backup stack.*
- virtual void `discardJacobianBackup` ()  
*Discard current jacobian backup.*
- virtual `JacobianWorkspace` & `getJacobianBackup` ()

### Static Public Attributes

- static const int `NoBounds` = Index >= 0 ? (Bound\_type == `BOUND_TYPE::LOWERUPPER` ? 2 : 1) : (Bound\_type == `BOUND_TYPE::LOWERUPPER` ? 2 \* Vertex::Dimension : Vertex::Dimension)  
*Store number of bounds.*
- static const `BOUND_TYPE` `BoundType` = Bound\_type  
*Store type of bound (lower, upper, or both);.*
- static constexpr const int `NoVertices` = sizeof...(Vertices)
- static const int `Dimension` = D  
*Edge dimension as static member variable.*

### Protected Attributes

- Eigen::Matrix< double,  
`NoBounds`, 1 > `_bounds`  
*Store bounds: first lower and then upper bound, depending on `Bound_type` parameter.*
- `ValueVector` `_values` = ValueVector::Zero()  
*Actual cost/constraint value data object (initialized to zero).*
- const std::array< `VertexType`  
\*, `NoVertices` > `_vertices`

Container that stores all attached vertices.

- `std::array< int, NoVertices > _vertices_dim`  
Store dimensions for all vertices (`EdgeType::_vertices`, `calculateVertexDimensions()`)
- `JacobianWorkspace _jacobians`  
Block-Jacobians of the edge (see `JacobianWorkspace`).
- `HessianWorkspace _hessians`  
Block-Hessians of the edge (see `HessianWorkspace`)
- `BackupStackType`  
< `JacobianWorkspace` > `_jacob_backup`

### 16.7.1 Detailed Description

```
template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1>class teb::BoundConstraint<
Bound_type, Vertex, Bound_vars, Index >
```

Use this class to create bound constraints  $\mathbf{x}_{min} \leq \mathbf{x} \leq \mathbf{x}_{max}$

#### Remarks

Call `allocateMemory()` before using the edge for optimization.

**Todo** Maybe use a special representation or zero Hessians to avoid processing a lot of zeros (same for other edges)

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

ConstraintEdgeUnary, ConstraintEdgeBinary, ConstraintEdgeMulti, ObjectiveEdgeUnary, ObjectiveEdgeBinary, ObjectiveEdgeMulti

#### Template Parameters

<i>Bound_type</i>	Lower or upper bound corresponding to BOUND_TYPE enum
<i>Vertex</i>	Type of the attached vertex.
<i>Bound_vars</i>	Specify whether a single state or all components of the <code>VertexType</code> should be bounded (See BOUND_VARS enum).
<i>Index</i>	If <code>Bound_vars==BOUND_VARS::SINGLE</code> then this Index specifies the element/component, otherwise set Index to -1.

Definition at line 55 of file bound\_constraints.h.

### 16.7.2 Member Typedef Documentation

16.7.2.1 `using teb::EdgeType::EdgeContainer = std::vector<EdgeType*> [inherited]`

Definition at line 114 of file graph.h.

16.7.2.2 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1>
using teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::ValueVector = typename
BaseEdge<NoBounds, FUNCT_TYPE::LINEAR, Vertex>::ValueVector`

Definition at line 70 of file bound\_constraints.h.

16.7.2.3 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]`

Definition at line 116 of file graph.h.

### 16.7.3 Constructor & Destructor Documentation

16.7.3.1 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1>  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::BoundConstraint ( Vertex & vertex )  
[inline]`

Parameters

<i>vertex</i>	Reference to the vertex that should be bounded
---------------	--

Definition at line 76 of file bound\_constraints.h.

### 16.7.4 Member Function Documentation

16.7.4.1 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices  
>::allocateMemory ( bool skip_hessian = false ) [inline],[virtual],[inherited]`

Remarks

Before calling, call `resizeVertexContainer()` edge and add all vertices (`setVertex()`) first.

**Todo** Implement automatic memory allocation instead calling this function manually.

Parameters

<i>skip_hessian</i>	If no hessian calculation is required by the solver skip memory allocation.
---------------------	---

Implements [teb::EdgeType](#).

Definition at line 84 of file base\_edge.h.

16.7.4.2 `virtual void teb::EdgeType::backupJacobian ( ) [inline],[virtual],[inherited]`

Definition at line 162 of file graph.h.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

16.7.4.3 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual void teb::BaseEdge< D, FuncType, Vertices  
>::calculateVertexDimensions ( ) [inline],[virtual],[inherited]`

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container `EdgeType::_vertices_dim`. This function is called within [BaseEdge::allocateMemory\(\)](#).

**Todo** Maybe implement static/constexpr version of collecting all dimensions since they are now at compile-time (maybe <http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array>

Definition at line 110 of file base\_edge.h.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, and `teb::BaseEdge< D, FuncType >::allocateMemory()`.

16.7.4.4 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian ( ) [inline],  
[virtual]`

Reimplemented from [teb::BaseEdge< D, FuncType, Vertices >](#).

Definition at line 163 of file `bound_constraints.h`.

References `teb::EdgeType::_hessians`, and `teb::HessianWorkspace::getWorkspace()`.

16.7.4.5 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian ( ) [inline],  
[virtual]`

Reimplemented from [teb::BaseEdge< D, FuncType, Vertices >](#).

Definition at line 125 of file `bound_constraints.h`.

References `teb::EdgeType::_jacobians`, `teb::JacobianWorkspace::getWorkspace()`, `teb::LOWER`, `teb::LOWERUPPER`, and `teb::SINGLE`.

16.7.4.6 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeValues ( ) [inline],  
[virtual]`

Reimplemented from [teb::BaseEdge< D, FuncType, Vertices >](#).

Definition at line 85 of file `bound_constraints.h`.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, `teb::BaseEdge< D, FuncType, Vertices >::_values`, `teb::BaseEdge< D, FuncType, Vertices >::_vertices`, `teb::LOWER`, `teb::LOWERUPPER`, and `teb::SINGLE`.

16.7.4.7 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual int teb::BaseEdge< D, FuncType, Vertices  
>::dimension ( ) const [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 51 of file `base_edge.h`.

16.7.4.8 `virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]`

Definition at line 171 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

16.7.4.9 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual FUNCT_TYPE teb::BaseEdge< D, FuncType,  
Vertices >::funcType ( ) const [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 69 of file `base_edge.h`.

16.7.4.10 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> const double*  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getBounds ( ) const [inline]`

See [setBounds\(\)](#) overloads for more information about the length. In summary:

- [BOUND\\_TYPE::LOWER](#) / [BOUND\\_TYPE::UPPER](#) && [BOUND\\_VARS::SINGLE](#) : 1
- [BOUND\\_TYPE::LOWER](#) / [BOUND\\_TYPE::UPPER](#) && [BOUND\\_VARS::ALL](#) : [Vertex::Dimension](#)

[BOUND\\_TYPE::LOWERUPPER](#) && [BOUND\\_VARS::SINGLE](#) : 2

- [BOUND\\_TYPE::LOWERUPPER](#) && [BOUND\\_VARS::ALL](#) :  $2 * \text{Vertex::Dimension}$

In case of [BOUND\\_TYPE::LOWERUPPER](#) first all lower bounds are stored and then all upper ones.

#### Returns

constant pointer to bound data

Definition at line 283 of file `bound_constraints.h`.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`.

**16.7.4.11** `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual void*  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData ( ) [inline],  
[virtual]`

This function overwrites the default interface to share custom data with edges. It avoids dynamic casting with known (!!!) template parameters. Cast the output to a [CustomBoundData](#) struct:

```
EdgeType* edge = new BoundConstraint<>(); // Do not forget to set bounds afterwards ...
CustomBoundData* bound_data = static_cast<CustomBoundData*>(edge->getCustomData());
// Do something with bound_data.
delete bound_data;
```

Do not forget to delete the `bound_data` object later

The bound data returned shares always the same size like `vertex->dimensionFree`.

All Bounds that are not considered in this class (like for `Index>0`) are set to -INF or INF depending on their type.

#### Returns

Void-pointer to [CustomBoundData](#) object on the heap.

Reimplemented from [teb::EdgeType](#).

Definition at line 308 of file `bound_constraints.h`.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, `teb::BaseEdge< D, FuncType, Vertices >::_vertices`, `teb::CustomBoundData::index`, `INF`, `teb::LOWER`, `teb::LOWERUPPER`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::NoBounds`, and `teb::UPPER`.

**16.7.4.12** `virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline],[virtual],  
[inherited]`

Definition at line 173 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

**16.7.4.13** `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual VertexType* teb::BaseEdge< D, FuncType,  
Vertices >::getVertex ( unsigned int idx ) [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 98 of file `base_edge.h`.



16.7.4.14 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) const [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 99 of file `base_edge.h`.

16.7.4.15 `virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline],[virtual],[inherited]`

Definition at line 159 of file `graph.h`.

References `teb::EdgeType::_hessians`.

16.7.4.16 `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline],[virtual],[inherited]`

Definition at line 160 of file `graph.h`.

References `teb::EdgeType::_hessians`.

16.7.4.17 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> virtual const bool teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::isBoundConstraint ( ) const [inline],[virtual]`

Reimplemented from [teb::EdgeType](#).

Definition at line 83 of file `bound_constraints.h`.

16.7.4.18 `virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline],[virtual],[inherited]`

Definition at line 157 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

16.7.4.19 `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline],[virtual],[inherited]`

Definition at line 158 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

16.7.4.20 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 101 of file `base_edge.h`.

16.7.4.21 `virtual void teb::EdgeType::restoreJacobian ( ) [inline],[virtual],[inherited]`

Definition at line 165 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

16.7.4.22 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds ( const double * bounds )  
[inline]`

Set an array of the bounds. The length depends on the class' template arguments BOUND\_TYPE Bound\_type and BOUND\_VARS Bound\_vars :

- [BOUND\\_TYPE::LOWER](#) / [BOUND\\_TYPE::UPPER](#) && [BOUND\\_VARS::SINGLE](#) : 1
- [BOUND\\_TYPE::LOWER](#) / [BOUND\\_TYPE::UPPER](#) && [BOUND\\_VARS::ALL](#) : Vertex::Dimension

Parameters

<i>bounds</i>	pointer to the bound-array.
---------------	-----------------------------

Definition at line 182 of file bound\_constraints.h.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, and `teb::LOWERUPPER`.

Referenced by `teb::TebController< p, q >::buildOptimizationGraph()`.

16.7.4.23 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds ( const Eigen::Ref< const  
Eigen::Matrix< double, NoBounds, 1 >> & bounds ) [inline]`

Set an array of the bounds. The length depends on the class' template arguments BOUND\_TYPE Bound\_type and BOUND\_VARS Bound\_vars :

- [BOUND\\_TYPE::LOWER](#) / [BOUND\\_TYPE::UPPER](#) && [BOUND\\_VARS::SINGLE](#) : 1
- [BOUND\\_TYPE::LOWER](#) / [BOUND\\_TYPE::UPPER](#) && [BOUND\\_VARS::ALL](#) : Vertex::Dimension

Parameters

<i>bounds</i>	Eigen Vector containing the bound.
---------------	------------------------------------

Definition at line 196 of file bound\_constraints.h.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, and `teb::LOWERUPPER`.

16.7.4.24 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds ( double bound ) [inline]`

Set a single bound. [BOUND\\_VARS::SINGLE](#) and `Index>=0` template parameter is required!

Parameters

<i>bound</i>	bound-value
--------------	-------------

Definition at line 207 of file bound\_constraints.h.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, `teb::LOWERUPPER`, and `teb::SINGLE`.

16.7.4.25 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void  
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds ( double lb, double ub )  
[inline]`

Set a single bound. [BOUND\\_VARS::SINGLE](#) and `Index>=0` template parameter are required!

## Parameters

<i>lb</i>	lower bound-value
<i>ub</i>	upper bound-value

Definition at line 219 of file `bound_constraints.h`.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, `teb::LOWERUPPER`, and `teb::SINGLE`.

```
16.7.4.26  template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void
            teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds ( const double * lb, const
            double * ub ) [inline]
```

Set an array of the bounds. The length depends on the class' template arguments `BOUND_TYPE Bound_type` and `BOUND_VARS Bound_vars` :

- `BOUND_TYPE::LOWERUPPER` && `BOUND_VARS::SINGLE` : 2
- `BOUND_TYPE::LOWERUPPER` && `BOUND_VARS::ALL` : `2*Vertex::Dimension`

## Parameters

<i>lb</i>	pointer to the lower bound-array.
<i>ub</i>	pointer to the upper bound-array.

Definition at line 235 of file `bound_constraints.h`.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, `teb::LOWERUPPER`, and `teb::SINGLE`.

```
16.7.4.27  template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> void
            teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds ( const Eigen::Ref< const
            Eigen::Matrix< double, NoBounds/2, 1 >> & lb, const Eigen::Ref< const Eigen::Matrix< double, NoBounds/2,
            1 >> & ub ) [inline]
```

Set an array of the bounds. The length depends on the class' template arguments `BOUND_TYPE Bound_type` and `BOUND_VARS Bound_vars` :

- `BOUND_TYPE::LOWERUPPER` && `BOUND_VARS::SINGLE` : 2
- `BOUND_TYPE::LOWERUPPER` && `BOUND_VARS::ALL` : `2*Vertex::Dimension`

## Parameters

<i>lb</i>	<code>Eigen::Vector [NoBounds x 1]</code> for the lower bound.
<i>ub</i>	<code>Eigen::Vector [NoBounds x 1]</code> for the upper bound.

Definition at line 262 of file `bound_constraints.h`.

References `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds`, `teb::LOWERUPPER`, and `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::NoBounds`.

```
16.7.4.28  template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const ValueVector& teb::BaseEdge< D,
            FuncType, Vertices >::values ( ) const [inline],[virtual],[inherited]
```

Definition at line 74 of file `base_edge.h`.

16.7.4.29 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline], [virtual], [inherited]`

Definition at line 75 of file base\_edge.h.

16.7.4.30 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 72 of file base\_edge.h.

16.7.4.31 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 73 of file base\_edge.h.

16.7.4.32 `virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]`

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References [teb::EdgeType::dimension\(\)](#), and [teb::EdgeType::valuesData\(\)](#).

16.7.4.33 `template<int D, FUNCT_TYPE FuncType, class... Vertices> virtual const std::array<VertexType*, NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual], [inherited]`

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base\_edge.h.

## 16.7.5 Member Data Documentation

16.7.5.1 `template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> Eigen::Matrix<double, NoBounds, 1> teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::_bounds [protected]`

Definition at line 377 of file bound\_constraints.h.

Referenced by [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >::computeValues\(\)](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >::getBounds\(\)](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >::getCustomData\(\)](#), and [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >::setBounds\(\)](#).

16.7.5.2 **HessianWorkspace** `teb::EdgeType::_hessians [protected], [inherited]`

Definition at line 178 of file graph.h.

Referenced by [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >::allocateMemory\(\)](#), [teb::BaseEdge< D, FuncType >::allocateMemory\(\)](#), [teb::EdgeMinimizeTime::computeHessian\(\)](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >::computeHessian\(\)](#), [teb::EdgeQuadraticForm< p, q >::computeHessian\(\)](#), [teb::BaseEdge< D, FuncType >::computeHessian\(\)](#), and [teb::EdgeType::hessians\(\)](#).

### 16.7.5.3 BackupStackType<JacobianWorkspace> teb::EdgeType::\_jacob\_backup [protected], [inherited]

Definition at line 180 of file graph.h.

Referenced by teb::EdgeType::backupJacobian(), teb::EdgeType::discardJacobianBackup(), teb::EdgeType::getJacobianBackup(), and teb::EdgeType::restoreJacobian().

### 16.7.5.4 JacobianWorkspace teb::EdgeType::\_jacobians [protected], [inherited]

Definition at line 173 of file graph.h.

Referenced by teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::EdgeType::backupJacobian(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::EdgeMinimizeTime::computeJacobian(), teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

### 16.7.5.5 template<int D, FUNCT\_TYPE FuncType, class... Vertices> ValueVector teb::BaseEdge< D, FuncType, Vertices >::\_values = ValueVector::Zero() [protected], [inherited]

Definition at line 139 of file base\_edge.h.

Referenced by teb::BaseEdge< D, FuncType >::computeJacobian(), teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index >::computeValues(), teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::values(), teb::BaseEdge< D, FuncType >::values(), teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::valuesData(), and teb::BaseEdge< D, FuncType >::valuesData().

### 16.7.5.6 template<int D, FUNCT\_TYPE FuncType, class... Vertices> const std::array<VertexType\*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::\_vertices [protected], [inherited]

Definition at line 141 of file base\_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions(), teb::BaseEdge< D, FuncType >::calculateVertexDimensions(), teb::BaseEdge< D, FuncType >::computeHessian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index >::computeValues(), teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index >::getCustomData(), teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::getVertex(), teb::BaseEdge< D, FuncType >::getVertex(), teb::BaseEdge< D, FuncType >::noVertices(), teb::BaseEdge< D, FuncType >::resizeVertexContainer(), teb::BaseEdge< D, FuncType >::setVertex(), teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::vertices(), and teb::BaseEdge< D, FuncType >::vertices().

### 16.7.5.7 template<int D, FUNCT\_TYPE FuncType, class... Vertices> std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::\_vertices\_dim [protected], [inherited]

Definition at line 142 of file base\_edge.h.

Referenced by teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::allocateMemory(), teb::BaseEdge< D, FuncType >::allocateMemory(), teb::BaseEdge< 1, FUNCT\_TYPE::LINEAR, TimeDiff >::calculateVertexDimensions(), and teb::BaseEdge< D, FuncType >::calculateVertexDimensions().

### 16.7.5.8 template<BOUND\_TYPE Bound\_type, typename Vertex, BOUND\_VARS Bound\_vars, int Index = -1> const BOUND\_TYPE teb::BoundConstraint< Bound\_type, Vertex, Bound\_vars, Index >::BoundType = Bound\_type [static]

Definition at line 66 of file bound\_constraints.h.

```
16.7.5.9  template<int D, FUNCT_TYPE FuncType, class... Vertices> const int teb::BaseEdge< D, FuncType, Vertices
>::Dimension = D  [static], [inherited]
```

Definition at line 50 of file base\_edge.h.

```
16.7.5.10  template<BOUND_TYPE Bound_type, typename Vertex, BOUND_VARS Bound_vars, int Index = -1> const int
teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::NoBounds = Index >= 0 ? (Bound_type
== BOUND_TYPE::LOWERUPPER ? 2 : 1) : (Bound_type == BOUND_TYPE::LOWERUPPER ? 2 *
Vertex::Dimension : Vertex::Dimension)  [static]
```

Definition at line 65 of file bound\_constraints.h.

Referenced by `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData()`, and `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds()`.

```
16.7.5.11  template<int D, FUNCT_TYPE FuncType, class... Vertices> constexpr const int teb::BaseEdge< D, FuncType,
Vertices >::NoVertices = sizeof...(Vertices)  [static], [inherited]
```

Definition at line 48 of file base\_edge.h.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::noVertices()`.

The documentation for this class was generated from the following file:

- [bound\\_constraints.h](#)

## 16.8 teb::Config Class Reference

Configurations for Controller and Solver classes.

```
#include <config.h>
```

### Classes

- struct [Optim](#)  
*Configurations related to the trajectory optimization.*
- struct [Teb](#)  
*Configurations related to the TEB algorithm.*
- struct [Utilities](#)  
*Configurations for helper miscellaneous and utilities.*

### Public Attributes

- struct [teb::Config::Teb](#) `teb`  
*Configurations related to the TEB algorithm.*
- struct [teb::Config::Optim](#) `optim`  
*Configurations related to the trajectory optimization.*
- struct [teb::Config::Utilities](#) `util`  
*Configurations for helper miscellaneous and utilities.*

### 16.8.1 Detailed Description

This class defines the main config for controller and solver classes.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### See Also

[BaseController](#) [TebController](#) [BaseSolver](#)

#### Examples:

[integrator\\_system1.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Definition at line 55 of file `config.h`.

### 16.8.2 Member Data Documentation

#### 16.8.2.1 `struct teb::Config::Optim` `teb::Config::optim`

#### Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.8.2.2 `struct teb::Config::Teb` `teb::Config::teb`

#### Examples:

[mobile\\_robot\\_teb.cpp](#).

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::TebController< p, q >::resetController()`, and `teb::TebController< p, q >::setupHorizon()`.

#### 16.8.2.3 `struct teb::Config::Utilities` `teb::Config::util`

The documentation for this class was generated from the following file:

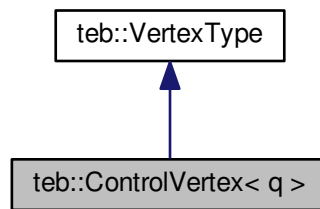
- [config.h](#)

## 16.9 `teb::ControlVertex< q >` Class Template Reference

Vertex that contains the `ControlVector`.

```
#include <teb_vertices.h>
```

Inheritance diagram for `teb::ControlVertex< q >`:



## Public Types

- using `ControlVector` = `Eigen::Matrix< double, q, 1 >`  
*Typedef for control input vector with q controls [q x 1].*
- using `ScalarType` = `Eigen::Matrix< double, 1, 1 >`  
*Typedef for a 1D-scalar value represented as Eigen::Matrix type [1 x 1].*
- using `FixedCtrls` = `Eigen::Matrix< bool, q, 1 >`  
*Typedef for logical mask that stores fixed and unfixed controls [q x 1].*
- using `BackupStackType` = `std::stack< ControlVector, std::vector< ControlVector, Eigen::aligned_allocator< ControlVector > > >`  
*Typedef for the backup stack.*
- using `VertexContainer` = `std::vector< VertexType * >`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*

## Public Member Functions

- `ControlVertex ()`  
*Default constructor: control vector initialized to zero.*
- `ControlVertex (const Eigen::Ref< const ControlVector > &controls)`  
*Construct ControlVertex with given control inputs.*
- `ControlVertex (const ControlVertex< q > &obj)`  
*Copy constructor.*
- virtual int `dimension () const`  
*Return number of elements/values/components stored in this vertex.*
- virtual int `dimensionFree () const`  
*Return only the dimension of free (unfixed) variables.*
- virtual void `plus (const VertexType *rhs)`  
*Define the increment for the vertex:  $x = x + rhs$ .*
- virtual void `plus (const double *rhs)`  
*Define the increment for the vertex:  $x = x + rhs$  (rhs[] with `dimension()`=p+q values)*
- virtual void `plusFree (const double *rhs)`  
*Define the increment for the vertex:  $x = x + rhs$  (But only add FREE variables, rhs[] with `dimensionFree()` values).*
- virtual void `setFree (const double *rhs)`  
*Overwrite all free variables with the values given by rhs (rhs[] with `dimensionFree()` values).*
- virtual void `getDataFree (double *target_vec) const`



- Return a copy of the free (unfixed) values as a single array ( length: `dimensionFree()` ).*

  - virtual const double & `getData` (unsigned int idx) const

*Return pointer to data with the index `idx` ).*
- const `ControlVector` & `controls` () const

*Get ControlVector (read-only).*
- `ControlVector` & `controls` ()

*Get ControlVector.*
- virtual void `setControls` (const Eigen::Ref< const `ControlVector` > &c)

*Set or update ControlVector.*
- virtual void `setControls` (double control)

*Set or update only first component of the ControlVector (useful for 1d controls).*
- const `FixedCtrls` & `fixed_ctrls` () const

*Get logical map that defines fixed and unfixed controls (fixed = true).*
- void `setFixedAll` (bool fixed)

*Set all controls of this object fixed or unfixed.*
- void `setFixedCtrl` (unsigned int ctrl\_idx, bool fixed)

*Set a single component of the ControlVector to fixed/unfixed.*
- void `setFixedCtrls` (bool fixed)

*Set all controls of the ControlVector to fixed/unfixed.*
- void `setFixedCtrls` (const Eigen::Ref< const `FixedCtrls` > &fixed\_ctrls)

*Set selected controls of the ControlVector to fixed/unfixed at once (Eigen version).*
- void `setFixedCtrls` (const bool \*fixed)

*Set selected components of the ControlVector to fixed/unfixed at once.*
- virtual bool `isFixedAll` () const

*return true, if ALL values of this vertex are fixed and therefore are not necessary for optimization.*
- bool `isFixedAny` () const

*return true, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.*
- virtual bool `isFixedComp` (int idx) const

*return true, if element/value/component with index `idx` of this vertex is fixed.*
- virtual void `push` ()

*This function should store all values into a internal backup stack.*
- virtual void `pop` ()

*This function should restore the previously stored values of the backup stack and remove them from the stack.*
- virtual void `top` ()

*This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.*
- virtual void `discardTop` ()

*This function should delete the previously made backup from the stack without restoring it.*
- virtual unsigned int `stackSize` () const

*This function should return the current size/number of backups of the backup stack.*
- `ControlVertex`< q > & `operator=` (const `ControlVertex`< q > &rhs)
- void `setOptVecIdx` (int opt\_vec\_idx)

*Update the position of the first value of this vertex inside the optimization vector / Hessian.*
- int `getOptVecIdx` () const

*Get the position of the first value of this Vertex inside the optimization vector / Hessian.*

## Static Public Attributes

- static const int `DimControls` = q
- Number of control inputs / Size of the control vector as static variable.*
- static const int `Dimension` = q
- Number of total values stored in this vertex (number of control input vector components).*

## Protected Attributes

- [ControlVector \\_controls](#)  
*Control input vector with  $q$  controls [ $q \times 1$ ].*
- [BackupStackType \\_backup](#)  
*backup stack (store and restore control input).*
- `int _dimension = q`  
*Store dimension for all values stored ( $q$  controls)*
- [FixedCtrls \\_fixed\\_ctrls](#) = FixedCtrls::Constant(false)  
*Mask that stores fixed and unfixed controls [ $q \times 1$ ] true: fixed, false: free.*
- `int _opt_vec_idx = -1`  
*Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))*

## Friends

- `bool operator== (const ControlVertex< q > &lhs, const ControlVertex< q > &rhs)`
- `bool operator!= (const ControlVertex< q > &lhs, const ControlVertex< q > &rhs)`
- `std::ostream & operator<< (std::ostream &os, const ControlVertex< q > &control)`  
*Print control vector using std::ostream.*

### 16.9.1 Detailed Description

`template<int q>class teb::ControlVertex< q >`

This class wraps the discrete control input vector  $\mathbf{u}_k \in \mathbb{R}^q$  into a vertex for the hyper-graph.

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

[StateVertex](#), [VertexType](#), [TimeDiff](#), [BaseEdge](#)

#### Template Parameters

$q$	Number of control inputs / Size of the control vector
-----	---

Definition at line 281 of file teb\_vertices.h.

### 16.9.2 Member Typedef Documentation

16.9.2.1 `template<int q> using teb::ControlVertex< q >::BackupStackType = std::stack<ControlVector, std::vector<ControlVector, Eigen::aligned_allocator<ControlVector> > >`

Definition at line 298 of file teb\_vertices.h.

16.9.2.2 `template<int q> using teb::ControlVertex< q >::ControlVector = Eigen::Matrix<double, q, 1>`

Definition at line 288 of file teb\_vertices.h.

16.9.2.3 `template<int q> using teb::ControlVertex< q >::FixedCtrls = Eigen::Matrix<bool, q, 1>`

Definition at line 292 of file teb\_vertices.h.

16.9.2.4 `template<int q> using teb::ControlVertex< q >::ScalarType = Eigen::Matrix<double, 1, 1>`

Definition at line 290 of file teb\_vertices.h.

16.9.2.5 `using teb::VertexType::VertexContainer = std::vector<VertexType*> [inherited]`

Definition at line 33 of file graph.h.

### 16.9.3 Constructor & Destructor Documentation

16.9.3.1 `template<int q> teb::ControlVertex< q >::ControlVertex ( ) [inline]`

Definition at line 306 of file teb\_vertices.h.

16.9.3.2 `template<int q> teb::ControlVertex< q >::ControlVertex ( const Eigen::Ref< const ControlVector > & controls ) [inline]`

#### Parameters

<i>controls</i>	Eigen::Vector with <i>q</i> control inputs.
-----------------	---

Definition at line 312 of file teb\_vertices.h.

16.9.3.3 `template<int q> teb::ControlVertex< q >::ControlVertex ( const ControlVertex< q > & obj ) [inline]`

#### Parameters

<i>obj</i>	Object of type <a href="#">ControlVertex</a>
------------	--

Definition at line 318 of file teb\_vertices.h.

References `teb::ControlVertex< q >::_fixed_ctrls`, and `teb::ControlVertex< q >::fixed_ctrls()`.

### 16.9.4 Member Function Documentation

16.9.4.1 `template<int q> const ControlVector& teb::ControlVertex< q >::controls ( ) const [inline]`

#### Returns

control vector [*q* x 1]

Definition at line 397 of file teb\_vertices.h.

References `teb::ControlVertex< q >::_controls`.

Referenced by `teb::ControlVertex< q >::operator=()`, and `teb::ControlVertex< q >::plus()`.

16.9.4.2 `template<int q> ControlVector& teb::ControlVertex< q >::controls ( ) [inline]`

#### Returns

control vector [*q* x 1]

Definition at line 398 of file teb\_vertices.h.

References `teb::ControlVertex< q >::_controls`.

**16.9.4.3** `template<int q> virtual int teb::ControlVertex< q >::dimension ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 327 of file `teb_vertices.h`.

**16.9.4.4** `template<int q> virtual int teb::ControlVertex< q >::dimensionFree ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 329 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`.

**16.9.4.5** `template<int q> virtual void teb::ControlVertex< q >::discardTop ( ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 456 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_backup`.

**16.9.4.6** `template<int q> const FixedCtrls& teb::ControlVertex< q >::fixed_ctrls ( ) const [inline]`

Returns

logical map ( $q \times 1$  vector of booleans)

Definition at line 401 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`.

Referenced by `teb::ControlVertex< q >::ControlVertex()`, `teb::ControlVertex< q >::operator=()`, and `teb::ControlVertex< q >::setFixedCtrls()`.

**16.9.4.7** `template<int q> virtual const double& teb::ControlVertex< q >::getData ( unsigned int idx ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 388 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`.

**16.9.4.8** `template<int q> virtual void teb::ControlVertex< q >::getDataFree ( double * target_vec ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 375 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`, and `teb::ControlVertex< q >::isFixedComp()`.

**16.9.4.9** `int teb::VertexType::getOptVecIdx ( ) const [inline],[inherited]`

**Returns**

position of the first element of this vertex.

Definition at line 59 of file graph.h.

References [teb::VertexType::\\_opt\\_vec\\_idx](#).

Referenced by [teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian\(\)](#), [teb::SolverLevenbergMarquardtEigenDense::buildJacobian\(\)](#), [teb::SolverLevenbergMarquardtEigenSparse::buildJacobian\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient\(\)](#), [teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically\(\)](#), and [teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ\(\)](#).

**16.9.4.10** `template<int q> virtual bool teb::ControlVertex< q >::isFixedAll ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 438 of file teb\_vertices.h.

References [teb::ControlVertex< q >::\\_fixed\\_ctrls](#).

**16.9.4.11** `template<int q> bool teb::ControlVertex< q >::isFixedAny ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 439 of file teb\_vertices.h.

References [teb::ControlVertex< q >::\\_fixed\\_ctrls](#).

**16.9.4.12** `template<int q> virtual bool teb::ControlVertex< q >::isFixedComp ( int idx ) const [inline],[virtual]`

**Parameters**

<i>idx</i>	component index
------------	-----------------

Implements [teb::VertexType](#).

Definition at line 441 of file teb\_vertices.h.

References [teb::ControlVertex< q >::\\_fixed\\_ctrls](#).

Referenced by [teb::ControlVertex< q >::getDataFree\(\)](#), [teb::ControlVertex< q >::plusFree\(\)](#), and [teb::ControlVertex< q >::setFree\(\)](#).

**16.9.4.13** `template<int q> ControlVertex<q>& teb::ControlVertex< q >::operator= ( const ControlVertex< q > & rhs ) [inline]`

Definition at line 461 of file teb\_vertices.h.

References [teb::ControlVertex< q >::\\_controls](#), [teb::ControlVertex< q >::\\_fixed\\_ctrls](#), [teb::ControlVertex< q >::controls\(\)](#), and [teb::ControlVertex< q >::fixed\\_ctrls\(\)](#).

**16.9.4.14** `template<int q> virtual void teb::ControlVertex< q >::plus ( const VertexType * rhs ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 332 of file teb\_vertices.h.

References [teb::ControlVertex< q >::\\_controls](#), and [teb::ControlVertex< q >::controls\(\)](#).

**16.9.4.15** `template<int q> virtual void teb::ControlVertex< q >::plus ( const double * rhs ) [inline],  
[virtual]`

Implements [teb::VertexType](#).

Definition at line 339 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`.

**16.9.4.16** `template<int q> virtual void teb::ControlVertex< q >::plusFree ( const double * rhs ) [inline],  
[virtual]`

Implements [teb::VertexType](#).

Definition at line 345 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`, and `teb::ControlVertex< q >::isFixedComp()`.

**16.9.4.17** `template<int q> virtual void teb::ControlVertex< q >::pop ( ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 446 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_backup`, and `teb::ControlVertex< q >::top()`.

**16.9.4.18** `template<int q> virtual void teb::ControlVertex< q >::push ( ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 445 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_backup`, and `teb::ControlVertex< q >::_controls`.

**16.9.4.19** `template<int q> virtual void teb::ControlVertex< q >::setControls ( const Eigen::Ref< const ControlVector  
> & c ) [inline],[virtual]`

#### Parameters

<i>c</i>	control vector [p x 1]
----------	------------------------

Definition at line 399 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`.

**16.9.4.20** `template<int q> virtual void teb::ControlVertex< q >::setControls ( double control ) [inline],  
[virtual]`

#### Parameters

<i>control</i>	single control component.
----------------	---------------------------

Definition at line 400 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`.

**16.9.4.21** `template<int q> void teb::ControlVertex< q >::setFixedAll ( bool fixed ) [inline]`

## Parameters

<i>fixed</i>	set to <code>true</code> in order to fix all <i>q</i> controls.
--------------	---

Definition at line 407 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`.

**16.9.4.22** `template<int q> void teb::ControlVertex< q >::setFixedCtrl ( unsigned int ctrl_idx, bool fixed )` `[inline]`

## Parameters

<i>ctrl_idx</i>	index of the underlying ControlVector component that should be fixed.
<i>fixed</i>	set to <code>true</code> in order to fix the selcted control.

Definition at line 418 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`.

**16.9.4.23** `template<int q> void teb::ControlVertex< q >::setFixedCtrls ( bool fixed )` `[inline]`

## Parameters

<i>fixed</i>	set to <code>true</code> in order to fix all <i>q</i> controls.
--------------	---

Definition at line 424 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`.

**16.9.4.24** `template<int q> void teb::ControlVertex< q >::setFixedCtrls ( const Eigen::Ref< const FixedCtrls > & fixed_ctrls )` `[inline]`

## Parameters

<i>fixed_ctrls</i>	<code>Eigen::Vector</code> with <i>q</i> booleans in which each <code>true</code> sets the corresponding control of the ControlVector to fixed.
--------------------	---

Definition at line 430 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`, and `teb::ControlVertex< q >::fixed_ctrls()`.

**16.9.4.25** `template<int q> void teb::ControlVertex< q >::setFixedCtrls ( const bool * fixed )` `[inline]`

## Parameters

<i>fixed</i>	boolean array with <i>q</i> elements in which each <code>true</code> sets the corresponding control of the ControlVector to fixed.
--------------	--

Definition at line 436 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_fixed_ctrls`.

**16.9.4.26** `template<int q> virtual void teb::ControlVertex< q >::setFree ( const double * rhs )` `[inline]`,  
`[virtual]`

Implements [teb::VertexType](#).

Definition at line 360 of file `teb_vertices.h`.

References `teb::ControlVertex< q >::_controls`, and `teb::ControlVertex< q >::isFixedComp()`.

16.9.4.27 void teb::VertexType::setOptVecIdx ( int *opt\_vec\_idx* ) [inline],[inherited]



## Parameters

<i>opt_vec_idx</i>	index of the first value
--------------------	--------------------------

Definition at line 53 of file graph.h.

References `teb::VertexType::_opt_vec_idx`.

**16.9.4.28** `template<int q> virtual unsigned int teb::ControlVertex< q >::stackSize ( ) const [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 457 of file teb\_vertices.h.

References `teb::ControlVertex< q >::_backup`.

**16.9.4.29** `template<int q> virtual void teb::ControlVertex< q >::top ( ) [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 451 of file teb\_vertices.h.

References `teb::ControlVertex< q >::_backup`, and `teb::ControlVertex< q >::_controls`.

Referenced by `teb::ControlVertex< q >::pop()`.

## 16.9.5 Friends And Related Function Documentation

**16.9.5.1** `template<int q> bool operator!= ( const ControlVertex< q > & lhs, const ControlVertex< q > & rhs ) [friend]`

Definition at line 479 of file teb\_vertices.h.

**16.9.5.2** `template<int q> std::ostream& operator<< ( std::ostream & os, const ControlVertex< q > & control ) [friend]`

## Parameters

<i>os</i>	output stream object
<i>control</i>	specific <a href="#">ControlVertex</a> object

## Returns

output stream object including a formatted string containing control input vector.

Definition at line 489 of file teb\_vertices.h.

**16.9.5.3** `template<int q> bool operator== ( const ControlVertex< q > & lhs, const ControlVertex< q > & rhs ) [friend]`

Definition at line 474 of file teb\_vertices.h.

## 16.9.6 Member Data Documentation

**16.9.6.1** `template<int q> BackupStackType teb::ControlVertex< q >::_backup [protected]`

Definition at line 497 of file teb\_vertices.h.

Referenced by `teb::ControlVertex< q >::discardTop()`, `teb::ControlVertex< q >::pop()`, `teb::ControlVertex< q >::push()`, `teb::ControlVertex< q >::stackSize()`, and `teb::ControlVertex< q >::top()`.

**16.9.6.2** `template<int q> ControlVector teb::ControlVertex< q >::_controls` `[protected]`

Definition at line 496 of file `teb_vertices.h`.

Referenced by `teb::ControlVertex< q >::controls()`, `teb::ControlVertex< q >::getData()`, `teb::ControlVertex< q >::getDataFree()`, `teb::ControlVertex< q >::operator=()`, `teb::ControlVertex< q >::plus()`, `teb::ControlVertex< q >::plusFree()`, `teb::ControlVertex< q >::push()`, `teb::ControlVertex< q >::setControls()`, `teb::ControlVertex< q >::setFree()`, and `teb::ControlVertex< q >::top()`.

**16.9.6.3** `template<int q> int teb::ControlVertex< q >::_dimension = q` `[protected]`

Definition at line 499 of file `teb_vertices.h`.

**16.9.6.4** `template<int q> FixedCtrls teb::ControlVertex< q >::_fixed_ctrls = FixedCtrls::Constant(false)`  
`[protected]`

Definition at line 500 of file `teb_vertices.h`.

Referenced by `teb::ControlVertex< q >::ControlVertex()`, `teb::ControlVertex< q >::dimensionFree()`, `teb::ControlVertex< q >::fixed_ctrls()`, `teb::ControlVertex< q >::isFixedAll()`, `teb::ControlVertex< q >::isFixedAny()`, `teb::ControlVertex< q >::isFixedComp()`, `teb::ControlVertex< q >::operator=()`, `teb::ControlVertex< q >::setFixedAll()`, `teb::ControlVertex< q >::setFixedCtrl()`, and `teb::ControlVertex< q >::setFixedCtrls()`.

**16.9.6.5** `int teb::VertexType::_opt_vec_idx = -1` `[protected]`, `[inherited]`

Definition at line 69 of file `graph.h`.

Referenced by `teb::VertexType::getOptVecIdx()`, and `teb::VertexType::setOptVecIdx()`.

**16.9.6.6** `template<int q> const int teb::ControlVertex< q >::DimControls = q` `[static]`

Definition at line 284 of file `teb_vertices.h`.

**16.9.6.7** `template<int q> const int teb::ControlVertex< q >::Dimension = q` `[static]`

Definition at line 285 of file `teb_vertices.h`.

The documentation for this class was generated from the following file:

- [teb\\_vertices.h](#)

## 16.10 teb::CustomBoundData Struct Reference

Datastruct for custom bound data that can be queried from the [BoundConstraint](#) class.

```
#include <bound_constraints.h>
```

### Public Attributes

- `int index = -1`

- Index for the bound (-1 for multiple components)*
- int `no_lower` = 0  
*Number of lower bounds.*
- Eigen::VectorXd `lower`  
*[no\_lower x 1] lower bounds vector*
- int `no_upper` = 0  
*Number of upper bounds.*
- Eigen::VectorXd `upper`  
*[no\_upper x 1] upper bounds vector*
- `EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

### 16.10.1 Detailed Description

Definition at line 23 of file `bound_constraints.h`.

### 16.10.2 Member Data Documentation

#### 16.10.2.1 `teb::CustomBoundData::EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

Definition at line 31 of file `bound_constraints.h`.

#### 16.10.2.2 `int teb::CustomBoundData::index = -1`

Definition at line 25 of file `bound_constraints.h`.

Referenced by `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData()`.

#### 16.10.2.3 `Eigen::VectorXd teb::CustomBoundData::lower`

Definition at line 27 of file `bound_constraints.h`.

#### 16.10.2.4 `int teb::CustomBoundData::no_lower = 0`

Definition at line 26 of file `bound_constraints.h`.

#### 16.10.2.5 `int teb::CustomBoundData::no_upper = 0`

Definition at line 28 of file `bound_constraints.h`.

#### 16.10.2.6 `Eigen::VectorXd teb::CustomBoundData::upper`

Definition at line 29 of file `bound_constraints.h`.

The documentation for this struct was generated from the following file:

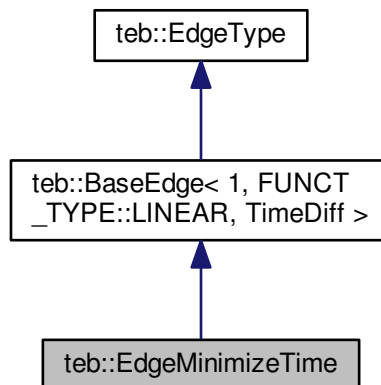
- [bound\\_constraints.h](#)

## 16.11 teb::EdgeMinimizeTime Class Reference

Objective edge: minimize  $\Delta T$ .

```
#include <common_teb_edges.h>
```

Inheritance diagram for teb::EdgeMinimizeTime:



### Public Types

- using [ValueVector](#) = Eigen::Matrix< double, D, 1 >  
*Typedef to represent the static Eigen::Vector of cost/constraint values.*
- using [EdgeContainer](#) = std::vector< [EdgeType](#) \* >  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*
- using [ValueVectorMap](#) = Eigen::Map< Eigen::VectorXd >  
*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

### Public Member Functions

- [EdgeMinimizeTime](#) ([TimeDiff](#) &dt\_vertex)  
*Construct edge and attach the corresponding [TimeDiff](#) vertex.*
- virtual void [computeValues](#) ()  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual void [computeJacobian](#) ()  
*Specification of the analytic block jacobian.*
- virtual void [computeHessian](#) ()  
*Specification of the analytic block hessian.*
- void [setData](#) (double data)  
*Set weight  $\sigma$  for the cost function.*
- virtual int [dimension](#) () const  
*Return edge dimension by function call.*
- virtual [FUNCT\\_TYPE](#) [funcType](#) () const  
*Return function typed given by template parameter *FuncType*.*
- virtual const double \* [valuesData](#) () const

- Return pointer to cost/constraint values [dimension() x 1] (read-only).*

  - virtual double \* [valuesData](#) ()

*Return pointer to cost/constraint values [dimension() x 1].*

  - virtual const [ValueVector](#) & [values](#) () const

*Return cost/constraint values as static Eigen::Vector (read-only).*

  - virtual [ValueVector](#) & [values](#) ()

*Return cost/constraint values as static Eigen::Vector.*

  - virtual void [allocateMemory](#) (bool skip\_hessian=false)

*Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.*

  - virtual [VertexType](#) \* [getVertex](#) (unsigned int idx)

*Access attached vertex with index idx.*

  - virtual const [VertexType](#) \* [getVertex](#) (unsigned int idx) const

*Access attached vertex with index idx (read-only).*

  - virtual unsigned int [noVertices](#) () const

*Return the number of attached vertices.*

  - virtual void [calculateVertexDimensions](#) ()

*Return the number of attached vertices.*

  - virtual const std::array  
< [VertexType](#) \*, [NoVertices](#) > & [vertices](#) () const

*Return vertex container of the edge.*

  - virtual const bool [isBoundConstraint](#) () const

*Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.*

  - virtual [ValueVectorMap](#) [valuesMap](#) ()

*Get cost/constraints values as Eigen type matrix.*

  - virtual void \* [getCustomData](#) ()

*Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.*

  - virtual [JacobianWorkspace](#) & [jacobians](#) ()

*Access the jacobian workspace of this edge.*

  - virtual const [JacobianWorkspace](#) & [jacobians](#) () const

*Access the jacobian workspace of this edge (read-only).*

  - virtual [HessianWorkspace](#) & [hessians](#) ()

*Access the hessian workspace of this edge.*

  - virtual const [HessianWorkspace](#) & [hessians](#) () const

*Access the jacobian workspace of this edge (read-only).*

  - virtual void [backupJacobian](#) ()

*Make a backup of the current jacobian block matrix.*

  - virtual void [restoreJacobian](#) ()

*Restore jacobian block matrix from the backup stack.*

  - virtual void [discardJacobianBackup](#) ()

*Discard current jacobian backup.*

  - virtual [JacobianWorkspace](#) & [getJacobianBackup](#) ()

## Static Public Attributes

- static constexpr const int [NoVertices](#)
  - static const int [Dimension](#)
- Edge dimension as static member variable.*

## Protected Attributes

- double `_data` = 1  
*Weight of the objective function.*
- [ValueVector](#) `_values`  
*Actual cost/constraint value data object (initialized to zero).*
- const std::array< [VertexType](#) \*, [NoVertices](#) > `_vertices`  
*Container that stores all attached vertices.*
- std::array< int, [NoVertices](#) > `_vertices_dim`  
*Store dimensions for all vertices (`EdgeType::_vertices`, [calculateVertexDimensions\(\)](#))*
- [JacobianWorkspace](#) `_jacobians`  
*Block-Jacobians of the edge (see [JacobianWorkspace](#)).*
- [HessianWorkspace](#) `_hessians`  
*Block-Hessians of the edge (see [HessianWorkspace](#)).*
- [BackupStackType](#) < [JacobianWorkspace](#) > `_jacob_backup`

### 16.11.1 Detailed Description

The following cost function is implemented:

$$v = \sigma \Delta T$$

$\sigma$  denotes the weight. Choosing  $(n-1)$  has performed fine in the past, since  $T = (n-1)\Delta T$ .

Set the weight  $\sigma$  using [setData\(\)](#).

Vertices required (connect the edge with the following vertices):

1. [TimeDiff](#)  $\Delta T$

#### Remarks

Call [allocateMemory\(\)](#) before using the edge for optimization.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

Definition at line 32 of file `common_teb_edges.h`.

### 16.11.2 Member Typedef Documentation

16.11.2.1 `using teb::EdgeType::EdgeContainer = std::vector<EdgeType*>` `[inherited]`

Definition at line 114 of file `graph.h`.

16.11.2.2 `using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1>`  
`[inherited]`

Definition at line 54 of file `base_edge.h`.

16.11.2.3 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd >` `[inherited]`

Definition at line 116 of file `graph.h`.

### 16.11.3 Constructor & Destructor Documentation

16.11.3.1 `teb::EdgeMinimizeTime::EdgeMinimizeTime ( TimeDiff & dt_vertex )` `[inline]`

#### Parameters

<i>dt_vertex</i>	Reference to the <a href="#">TimeDiff</a> vertex
------------------	--

Definition at line 40 of file `common_teb_edges.h`.

### 16.11.4 Member Function Documentation

16.11.4.1 `virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory ( bool skip_hessian = false )`  
`[inline], [virtual], [inherited]`

#### Remarks

Before calling, call `resizeVertexContainer()` edge and add all vertices (`setVertex()`) first.

**Todo** Implement automatic memory allocation instead calling this function manually.

#### Parameters

<i>skip_hessian</i>	If no hessian calculation is required by the solver skip memory allocation.
---------------------	---

Implements [teb::EdgeType](#).

Definition at line 84 of file `base_edge.h`.

References `teb::EdgeType::_hessians`, `teb::EdgeType::_jacobians`, `teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim`, `teb::JacobianWorkspace::allocate()`, `teb::HessianWorkspace::allocate()`, and `teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions()`.

16.11.4.2 `virtual void teb::EdgeType::backupJacobian ( )` `[inline], [virtual], [inherited]`

Definition at line 162 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

16.11.4.3 `virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( )` `[inline], [virtual], [inherited]`

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container `EdgeType::_vertices_dim`. This function is called within `BaseEdge::allocateMemory()`.

**Todo** Maybe implement static/constexpr version of collecting all dimensions since they are now at compile-time (maybe <http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array>).

Definition at line 110 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`, and `teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim`.

16.11.4.4 `virtual void teb::EdgeMinimizeTime::computeHessian ( )` `[inline], [virtual]`

Reimplemented from [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

Definition at line 59 of file `common_teb_edges.h`.

References `teb::EdgeType::_hessians`, and `teb::HessianWorkspace::getWorkspace()`.

**16.11.4.5** `virtual void teb::EdgeMinimizeTime::computeJacobian ( ) [inline],[virtual]`

Reimplemented from [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

Definition at line 52 of file `common_teb_edges.h`.

References `_data`, `teb::EdgeType::_jacobians`, and `teb::JacobianWorkspace::getWorkspace()`.

**16.11.4.6** `virtual void teb::EdgeMinimizeTime::computeValues ( ) [inline],[virtual]`

Reimplemented from [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

Definition at line 45 of file `common_teb_edges.h`.

References `_data`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::_values`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::_vertices`, and `teb::TimeDiff::dt()`.

**16.11.4.7** `virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 51 of file `base_edge.h`.

**16.11.4.8** `virtual void teb::EdgeType::discardJacobianBackup ( ) [inline],[virtual],[inherited]`

Definition at line 171 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

**16.11.4.9** `virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::funcType ( ) const [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 69 of file `base_edge.h`.

**16.11.4.10** `virtual void* teb::EdgeType::getCustomData ( ) [inline],[virtual],[inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 154 of file `graph.h`.

**16.11.4.11** `virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline],[virtual],[inherited]`

Definition at line 173 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

**16.11.4.12** `virtual VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 98 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.



**16.11.4.13** `virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) const` `[inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 99 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

**16.11.4.14** `virtual HessianWorkspace& teb::EdgeType::hessians ( )` `[inline], [virtual], [inherited]`

Definition at line 159 of file `graph.h`.

References `teb::EdgeType::_hessians`.

**16.11.4.15** `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const` `[inline], [virtual], [inherited]`

Definition at line 160 of file `graph.h`.

References `teb::EdgeType::_hessians`.

**16.11.4.16** `virtual const bool teb::EdgeType::isBoundConstraint ( ) const` `[inline], [virtual], [inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 124 of file `graph.h`.

**16.11.4.17** `virtual JacobianWorkspace& teb::EdgeType::jacobians ( )` `[inline], [virtual], [inherited]`

Definition at line 157 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

**16.11.4.18** `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const` `[inline], [virtual], [inherited]`

Definition at line 158 of file `graph.h`.

References `teb::EdgeType::_jacobians`.

**16.11.4.19** `virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const` `[inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 101 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::NoVertices`.

**16.11.4.20** `virtual void teb::EdgeType::restoreJacobian ( )` `[inline], [virtual], [inherited]`

Definition at line 165 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.11.4.21** `void teb::EdgeMinimizeTime::setData ( double data ) [inline]`

Definition at line 64 of file `common_teb_edges.h`.

References `_data`.

Referenced by `teb::TebController< p, q >::buildOptimizationGraph()`.

**16.11.4.22** `virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline],  
[virtual],[inherited]`

Definition at line 74 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.11.4.23** `virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline],[virtual],  
[inherited]`

Definition at line 75 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.11.4.24** `virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline],  
[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 72 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.11.4.25** `virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline],[virtual],  
[inherited]`

Implements [teb::EdgeType](#).

Definition at line 73 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_values`.

**16.11.4.26** `virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline],[virtual],[inherited]`

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file `graph.h`.

References `teb::EdgeType::dimension()`, and `teb::EdgeType::valuesData()`.

**16.11.4.27** `virtual const std::array<VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( )  
const [inline],[virtual],[inherited]`

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file `base_edge.h`.

References `teb::BaseEdge< D, FuncType, Vertices >::_vertices`.

### 16.11.5 Member Data Documentation

**16.11.5.1** `double teb::EdgeMinimizeTime::_data = 1` `[protected]`

Definition at line 67 of file `common_teb_edges.h`.

Referenced by `computeJacobian()`, `computeValues()`, and `setData()`.

**16.11.5.2** `HessianWorkspace teb::EdgeType::_hessians` `[protected]`, `[inherited]`

Definition at line 178 of file `graph.h`.

Referenced by `teb::BaseEdge< D, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `computeHessian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian()`, `teb::EdgeQuadraticForm< p, q >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, and `teb::EdgeType::hessians()`.

**16.11.5.3** `BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup` `[protected]`, `[inherited]`

Definition at line 180 of file `graph.h`.

Referenced by `teb::EdgeType::backupJacobian()`, `teb::EdgeType::discardJacobianBackup()`, `teb::EdgeType::getJacobianBackup()`, and `teb::EdgeType::restoreJacobian()`.

**16.11.5.4** `JacobianWorkspace teb::EdgeType::_jacobians` `[protected]`, `[inherited]`

Definition at line 173 of file `graph.h`.

Referenced by `teb::BaseEdge< D, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::EdgeType::backupJacobian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian()`, `teb::EdgeQuadraticForm< p, q >::computeJacobian()`, `teb::BaseEdge< D, FuncType >::computeJacobian()`, `teb::EdgeType::jacobians()`, and `teb::EdgeType::restoreJacobian()`.

**16.11.5.5** `ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values` `[protected]`, `[inherited]`

Definition at line 139 of file `base_edge.h`.

Referenced by `computeValues()`.

**16.11.5.6** `const std::array<VertexType*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices` `[protected]`, `[inherited]`

Definition at line 141 of file `base_edge.h`.

Referenced by `computeValues()`.

**16.11.5.7** `std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim` `[protected]`, `[inherited]`

Definition at line 142 of file `base_edge.h`.

**16.11.5.8** `const int teb::BaseEdge< D, FuncType, Vertices >::Dimension` `[static]`, `[inherited]`

Definition at line 50 of file `base_edge.h`.

16.11.5.9 constexpr const int **teb::BaseEdge**< D, FuncType, Vertices >::NoVertices [static],[inherited]

Definition at line 48 of file base\_edge.h.

The documentation for this class was generated from the following file:

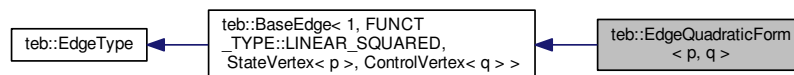
- [common\\_teb\\_edges.h](#)

## 16.12 teb::EdgeQuadraticForm< p, q > Class Template Reference

Objective edge: quadratic form for states and control input.

```
#include <common_teb_edges.h>
```

Inheritance diagram for teb::EdgeQuadraticForm< p, q >:



### Public Types

- using [ValueVector](#) = Eigen::Matrix< double, D, 1 >  
*Typedef to represent the static Eigen::Vector of cost/constraint values.*
- using [EdgeContainer](#) = std::vector< [EdgeType](#) \* >  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*
- using [ValueVectorMap](#) = Eigen::Map< Eigen::VectorXd >  
*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

### Public Member Functions

- [EdgeQuadraticForm](#) ([StateVertex](#)< p > &state\_vertex, [ControlVertex](#)< q > &control\_vertex)  
*Construct edge and attach the corresponding state and control vertex.*
- virtual void [computeValues](#) ()  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual void [computeJacobian](#) ()  
*Specification of the analytic block jacobian.*
- virtual void [computeHessian](#) ()  
*Specification of the analytic block hessian.*
- void [setWeights](#) (const Eigen::DiagonalMatrix< double, p > &Q, const Eigen::DiagonalMatrix< double, q > &R)  
*Set weights **Q** and **R** for the cost function.*
- void [setWeights](#) (const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> &Q, const Eigen::Ref< const Eigen::Matrix< double, q, 1 >> &R)  
*Set diagonal weights **Q** and **R** for the cost function.*
- void [setWeights](#) (double Q, double R)  
*Set weights **Q** and **R** for the cost function using uniform diagonal elements.*
- void [setReference](#) (const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> &xf)  
*Set reference state.*

- virtual int [dimension](#) () const  
*Return edge dimension by function call.*
- virtual [FUNCT\\_TYPE](#) [functType](#) () const  
*Return function typed given by template parameter FuncType.*
- virtual const double \* [valuesData](#) () const  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1] (read-only).*
- virtual double \* [valuesData](#) ()  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1].*
- virtual const [ValueVector](#) & [values](#) () const  
*Return cost/constraint values as static Eigen::Vector (read-only).*
- virtual [ValueVector](#) & [values](#) ()  
*Return cost/constraint values as static Eigen::Vector.*
- virtual void [allocateMemory](#) (bool skip\_hessian=false)  
*Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.*
- virtual [VertexType](#) \* [getVertex](#) (unsigned int idx)  
*Access attached vertex with index [idx](#).*
- virtual const [VertexType](#) \* [getVertex](#) (unsigned int idx) const  
*Access attached vertex with index [idx](#) (read-only).*
- virtual unsigned int [noVertices](#) () const  
*Return the number of attached vertices.*
- virtual void [calculateVertexDimensions](#) ()  
*Return the number of attached vertices.*
- virtual const std::array  
< [VertexType](#) \*, [NoVertices](#) > & [vertices](#) () const  
*Return vertex container of the edge.*
- virtual const bool [isBoundConstraint](#) () const  
*Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.*
- virtual [ValueVectorMap](#) [valuesMap](#) ()  
*Get cost/constraints values as Eigen type matrix.*
- virtual void \* [getCustomData](#) ()  
*Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.*
- virtual [JacobianWorkspace](#) & [jacobians](#) ()  
*Access the jacobian workspace of this edge.*
- virtual const [JacobianWorkspace](#) & [jacobians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual [HessianWorkspace](#) & [hessians](#) ()  
*Access the hessian workspace of this edge.*
- virtual const [HessianWorkspace](#) & [hessians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual void [backupJacobian](#) ()  
*Make a backup of the current jacobian block matrix.*
- virtual void [restoreJacobian](#) ()  
*Restore jacobian block matrix from the backup stack.*
- virtual void [discardJacobianBackup](#) ()  
*Discard current jacobian backup.*
- virtual [JacobianWorkspace](#) & [getJacobianBackup](#) ()

## Static Public Attributes

- static constexpr const int [NoVertices](#)
- static const int [Dimension](#)

*Edge dimension as static member variable.*

## Protected Types

- typedef Eigen::Matrix< double,  
p, 1 > [StateVector](#)

*Typedef for a StateVector type (required here to avoid gcc bug within the next line)*

## Protected Attributes

- Eigen::DiagonalMatrix< double, p > [\\_Q](#)  
*Weight for the state vector  $\mathbf{x}_k$ .*
- Eigen::DiagonalMatrix< double, q > [\\_R](#)  
*Weight for the control input  $\mathbf{u}_k$ .*
- [StateVector](#) [\\_ref\\_state](#) = StateVector::Zero()  
*Store reference state (final/goal state)*
- [ValueVector](#) [\\_values](#)  
*Actual cost/constraint value data object (initialized to zero).*
- const std::array< [VertexType](#)  
\*, [NoVertices](#) > [\\_vertices](#)  
*Container that stores all attached vertices.*
- std::array< int, [NoVertices](#) > [\\_vertices\\_dim](#)  
*Store dimensions for all vertices (EdgeType::\_vertices, [calculateVertexDimensions\(\)](#))*
- [JacobianWorkspace](#) [\\_jacobians](#)  
*Block-Jacobians of the edge (see [JacobianWorkspace](#)).*
- [HessianWorkspace](#) [\\_hessians](#)  
*Block-Hessians of the edge (see [HessianWorkspace](#)).*
- [BackupStackType](#)  
< [JacobianWorkspace](#) > [\\_jacob\\_backup](#)

### 16.12.1 Detailed Description

```
template<int p, int q>class teb::EdgeQuadraticForm< p, q >
```

The following cost function is implemented:

$$J_k = (\mathbf{x}_k - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{ref}) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k$$

$\mathbf{Q}$  denotes the weight for the state vector  $\mathbf{x}_k$ . and  $\mathbf{R}$  denotes the weight for the control input vector  $\mathbf{u}_k$ .

If this edge is added to all samples (states and control inputs) while composing the hyper-graph, it generates the following cost-function for the underlying optimization problem:

$$J = (\mathbf{x}_n - \mathbf{x}_{ref})^T \mathbf{Q}_n (\mathbf{x}_n - \mathbf{x}_{ref}) + \sum_{k=0}^{n-1} ((\mathbf{x}_k - \mathbf{x}_{ref})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{ref}) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k)$$

The weight for the final constraint (and for all others) can be chosen by specifying different weights. Set the weights  $\mathbf{Q}$  and  $\mathbf{R}$  using [setWeights\(\)](#).

Vertices required (connect the edge with the following vertices):

1. [StateVertex](#)  $\mathbf{x}_k$
2. [ControlVertex](#)  $\mathbf{u}_k$

If you only want to apply the quadratic form to either only states or control inputs, set `q=0` or `p=0`. A specialized template is provided.

**Todo** Currently the Levenberg-Marquardt solver take the cost function squared by itself, therefore we need to implement the sqrt here: For other solvers this cost function will be problematic without taking the square.

#### Remarks

We decided to allow only diagonal matrices for the weights at the moment in order to improve speed. Call [allocateMemory\(\)](#) before using the edge for optimization.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

Definition at line 168 of file `common_teb_edges.h`.

### 16.12.2 Member Typedef Documentation

16.12.2.1 `using teb::EdgeType::EdgeContainer = std::vector<EdgeType*> [inherited]`

Definition at line 114 of file `graph.h`.

16.12.2.2 `template<int p, int q> typedef Eigen::Matrix<double,p,1> teb::EdgeQuadraticForm< p, q >::StateVector [protected]`

Definition at line 265 of file `common_teb_edges.h`.

16.12.2.3 `using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1> [inherited]`

Definition at line 54 of file `base_edge.h`.

16.12.2.4 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]`

Definition at line 116 of file `graph.h`.

### 16.12.3 Constructor & Destructor Documentation

16.12.3.1 `template<int p, int q> teb::EdgeQuadraticForm< p, q >::EdgeQuadraticForm ( StateVertex< p > & state_vertex, ControlVertex< q > & control_vertex ) [inline]`

#### Parameters

<i>state_vertex</i>	Reference to the <a href="#">StateVertex</a>
---------------------	--

<i>control_vertex</i>	Reference to the <a href="#">ControlVertex</a>
-----------------------	--

Definition at line 177 of file common\_teb\_edges.h.

## 16.12.4 Member Function Documentation

**16.12.4.1** `virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory ( bool skip_hessian = false )`  
`[inline],[virtual],[inherited]`

### Remarks

Before calling, call `resizeVertexContainer()` edge and add all vertices (`setVertex()`) first.

**Todo** Implement automatic memory allocation instead calling this function manually.

### Parameters

<i>skip_hessian</i>	If no hessian calculation is required by the solver skip memory allocation.
---------------------	---

Implements [teb::EdgeType](#).

Definition at line 84 of file base\_edge.h.

**16.12.4.2** `virtual void teb::EdgeType::backupJacobian ( )` `[inline],[virtual],[inherited]`

Definition at line 162 of file graph.h.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.12.4.3** `virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( )` `[inline],[virtual],[inherited]`

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container `EdgeType::_vertices_dim`. This function is called within `BaseEdge::allocateMemory()`.

**Todo** Maybe implement static/constexpr version of collecting all dimensions since they are known at compile-time (maybe <http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array>

Definition at line 110 of file base\_edge.h.

**16.12.4.4** `template<int p, int q> virtual void teb::EdgeQuadraticForm< p, q >::computeHessian ( )` `[inline],[virtual]`

The hessian w.r.t. a symmetric weight matrix **A** is defined by

$$\frac{\partial^2}{\partial \mathbf{x}, \partial \mathbf{x}} \mathbf{x}^T \mathbf{A} \mathbf{x} = 2\mathbf{A}$$

Applying this formula to the states and control inputs with respect to weights **Q** and **R** implies a block diagonal matrix (even a diagonal matrix, since the blocks are diagonal as well).

Reimplemented from [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q >>](#).

Definition at line 217 of file common\_teb\_edges.h.

References `teb::EdgeType::_hessians`, `teb::EdgeQuadraticForm< p, q >::_Q`, `teb::EdgeQuadraticForm< p, q >::_R`, and `teb::HessianWorkspace::getWorkspace()`.



16.12.4.5 `template<int p, int q> virtual void teb::EdgeQuadraticForm< p, q >::computeJacobian ( ) [inline], [virtual]`

It is:

$$\frac{\partial}{\partial \mathbf{x}} \mathbf{x}^T \mathbf{A} \mathbf{x} = \mathbf{x}^T \mathbf{A} + \mathbf{A}^T = 2\mathbf{x}^T \mathbf{A}$$

The last step is allowed since  $\mathbf{A}$  is symmetric. This computation is now performed for both states and control inputs w.r.t  $\mathbf{Q}$  and  $\mathbf{R}$ .

Reimplemented from [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#).

Definition at line 200 of file `common_teb_edges.h`.

References `teb::EdgeType::_jacobians`, `teb::EdgeQuadraticForm< p, q >::_Q`, `teb::EdgeQuadraticForm< p, q >::_R`, and `teb::JacobianWorkspace::getWorkspace()`.

16.12.4.6 `template<int p, int q> virtual void teb::EdgeQuadraticForm< p, q >::computeValues ( ) [inline], [virtual]`

Reimplemented from [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#).

Definition at line 182 of file `common_teb_edges.h`.

References `teb::EdgeQuadraticForm< p, q >::_Q`, `teb::EdgeQuadraticForm< p, q >::_R`, `teb::EdgeQuadraticForm< p, q >::_ref_state`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >::_values`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >::_vertices`, and `teb::StateVertex< p >::states()`.

16.12.4.7 `virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 51 of file `base_edge.h`.

16.12.4.8 `virtual void teb::EdgeType::discardJacobianBackup ( ) [inline], [virtual], [inherited]`

Definition at line 171 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

16.12.4.9 `virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::funcType ( ) const [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 69 of file `base_edge.h`.

16.12.4.10 `virtual void* teb::EdgeType::getCustomData ( ) [inline], [virtual], [inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 154 of file `graph.h`.

**16.12.4.11** `virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline],[virtual],[inherited]`

Definition at line 173 of file graph.h.

References `teb::EdgeType::_jacob_backup`.

**16.12.4.12** `virtual VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 98 of file base\_edge.h.

**16.12.4.13** `virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) const [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 99 of file base\_edge.h.

**16.12.4.14** `virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline],[virtual],[inherited]`

Definition at line 159 of file graph.h.

References `teb::EdgeType::_hessians`.

**16.12.4.15** `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline],[virtual],[inherited]`

Definition at line 160 of file graph.h.

References `teb::EdgeType::_hessians`.

**16.12.4.16** `virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline],[virtual],[inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 124 of file graph.h.

**16.12.4.17** `virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline],[virtual],[inherited]`

Definition at line 157 of file graph.h.

References `teb::EdgeType::_jacobians`.

**16.12.4.18** `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline],[virtual],[inherited]`

Definition at line 158 of file graph.h.

References `teb::EdgeType::_jacobians`.

16.12.4.19 **virtual unsigned int** **teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const** [inline],  
[virtual],[inherited]

Implements [teb::EdgeType](#).

Definition at line 101 of file `base_edge.h`.

16.12.4.20 **virtual void** **teb::EdgeType::restoreJacobian ( )** [inline],[virtual],[inherited]

Definition at line 165 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

16.12.4.21 **template<int p, int q> void** **teb::EdgeQuadraticForm< p, q >::setReference ( const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> & xf )** [inline]

Definition at line 246 of file `common_teb_edges.h`.

References `teb::EdgeQuadraticForm< p, q >::_ref_state`.

Referenced by `teb::TebController< p, q >::buildOptimizationGraph()`.

16.12.4.22 **template<int p, int q> void** **teb::EdgeQuadraticForm< p, q >::setWeights ( const Eigen::DiagonalMatrix< double, p > & Q, const Eigen::DiagonalMatrix< double, q > & R )** [inline]

Definition at line 225 of file `common_teb_edges.h`.

References `teb::EdgeQuadraticForm< p, q >::_Q`, and `teb::EdgeQuadraticForm< p, q >::_R`.

Referenced by `teb::TebController< p, q >::buildOptimizationGraph()`.

16.12.4.23 **template<int p, int q> void** **teb::EdgeQuadraticForm< p, q >::setWeights ( const Eigen::Ref< const Eigen::Matrix< double, p, 1 >> & Q, const Eigen::Ref< const Eigen::Matrix< double, q, 1 >> & R )** [inline]

Definition at line 232 of file `common_teb_edges.h`.

References `teb::EdgeQuadraticForm< p, q >::_Q`, and `teb::EdgeQuadraticForm< p, q >::_R`.

16.12.4.24 **template<int p, int q> void** **teb::EdgeQuadraticForm< p, q >::setWeights ( double Q, double R )** [inline]

Definition at line 239 of file `common_teb_edges.h`.

References `teb::EdgeQuadraticForm< p, q >::_Q`, and `teb::EdgeQuadraticForm< p, q >::_R`.

16.12.4.25 **virtual const ValueVector&** **teb::BaseEdge< D, FuncType, Vertices >::values ( ) const** [inline],  
[virtual],[inherited]

Definition at line 74 of file `base_edge.h`.

16.12.4.26 **virtual ValueVector&** **teb::BaseEdge< D, FuncType, Vertices >::values ( )** [inline],[virtual],  
[inherited]

Definition at line 75 of file `base_edge.h`.

**16.12.4.27** `virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 72 of file base\_edge.h.

**16.12.4.28** `virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline], [virtual], [inherited]`

Implements [teb::EdgeType](#).

Definition at line 73 of file base\_edge.h.

**16.12.4.29** `virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline], [virtual], [inherited]`

return Eigen::Vector type that maps to the actual cost/constraints values

Definition at line 150 of file graph.h.

References [teb::EdgeType::dimension\(\)](#), and [teb::EdgeType::valuesData\(\)](#).

**16.12.4.30** `virtual const std::array<VertexType*, NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( ) const [inline], [virtual], [inherited]`

Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base\_edge.h.

## 16.12.5 Member Data Documentation

**16.12.5.1** `HessianWorkspace teb::EdgeType::_hessians [protected], [inherited]`

Definition at line 178 of file graph.h.

Referenced by [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >::allocateMemory\(\)](#), [teb::BaseEdge< D, FuncType >::allocateMemory\(\)](#), [teb::EdgeMinimizeTime::computeHessian\(\)](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >::computeHessian\(\)](#), [teb::EdgeQuadraticForm< p, q >::computeHessian\(\)](#), [teb::BaseEdge< D, FuncType >::computeHessian\(\)](#), and [teb::EdgeType::hessians\(\)](#).

**16.12.5.2** `BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected], [inherited]`

Definition at line 180 of file graph.h.

Referenced by [teb::EdgeType::backupJacobian\(\)](#), [teb::EdgeType::discardJacobianBackup\(\)](#), [teb::EdgeType::getJacobianBackup\(\)](#), and [teb::EdgeType::restoreJacobian\(\)](#).

**16.12.5.3** `JacobianWorkspace teb::EdgeType::_jacobians [protected], [inherited]`

Definition at line 173 of file graph.h.

Referenced by [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >::allocateMemory\(\)](#), [teb::BaseEdge< D, FuncType >::allocateMemory\(\)](#), [teb::EdgeType::backupJacobian\(\)](#), [teb::BaseEdge< D, FuncType >::computeHessian\(\)](#), [teb::EdgeMinimizeTime::computeJacobian\(\)](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars,](#)

Index >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::BaseEdge< D, FuncType >::computeJacobian(), teb::EdgeType::jacobians(), and teb::EdgeType::restoreJacobian().

**16.12.5.4** `template<int p, int q> Eigen::DiagonalMatrix<double,p> teb::EdgeQuadraticForm< p, q >::_Q`  
[protected]

Definition at line 262 of file common\_teb\_edges.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeValues(), and teb::EdgeQuadraticForm< p, q >::setWeights().

**16.12.5.5** `template<int p, int q> Eigen::DiagonalMatrix<double,q> teb::EdgeQuadraticForm< p, q >::_R`  
[protected]

Definition at line 263 of file common\_teb\_edges.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeHessian(), teb::EdgeQuadraticForm< p, q >::computeJacobian(), teb::EdgeQuadraticForm< p, q >::computeValues(), and teb::EdgeQuadraticForm< p, q >::setWeights().

**16.12.5.6** `template<int p, int q> StateVector teb::EdgeQuadraticForm< p, q >::_ref_state = StateVector::Zero()`  
[protected]

Definition at line 266 of file common\_teb\_edges.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeValues(), and teb::EdgeQuadraticForm< p, q >::setReference().

**16.12.5.7** `ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values` [protected],[inherited]

Definition at line 139 of file base\_edge.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeValues().

**16.12.5.8** `const std::array<VertexType*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices`  
[protected],[inherited]

Definition at line 141 of file base\_edge.h.

Referenced by teb::EdgeQuadraticForm< p, q >::computeValues().

**16.12.5.9** `std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim` [protected],[inherited]

Definition at line 142 of file base\_edge.h.

**16.12.5.10** `const int teb::BaseEdge< D, FuncType, Vertices >::Dimension` [static],[inherited]

Definition at line 50 of file base\_edge.h.

**16.12.5.11** `constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices` [static],[inherited]

Definition at line 48 of file base\_edge.h.

The documentation for this class was generated from the following file:

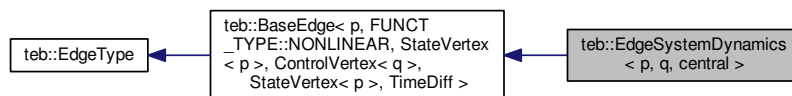
- [common\\_teb\\_edges.h](#)

## 16.13 `teb::EdgeSystemDynamics< p, q, central >` Class Template Reference

Equality constraint edge for satisfying continuous system dynamics specified by an [SystemDynamics](#) object.

```
#include <base_edge_dynamics.h>
```

Inheritance diagram for `teb::EdgeSystemDynamics< p, q, central >`:



### Public Types

- using [ValueVector](#) = `Eigen::Matrix< double, D, 1 >`  
*Typedef to represent the static Eigen::Vector of cost/constraint values.*
- using [EdgeContainer](#) = `std::vector< EdgeType * >`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*
- using [ValueVectorMap](#) = `Eigen::Map< Eigen::VectorXd >`  
*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

### Public Member Functions

- [EdgeSystemDynamics](#) ([SystemDynamics](#)< p, q > \*system, [StateVertex](#)< p > &px1, [ControlVertex](#)< q > &pu1, [StateVertex](#)< p > &px2, [TimeDiff](#) &pdt)  
*Constructor that requires a pointer to a [SystemDynamics](#) object.*
- virtual void [computeValues](#) ()  
*Actual cost function or constraint function. For constraints it is [computeValues](#)() <= 0.*
- void [setSystemDynamics](#) ([SystemDynamics](#)< p, q > \*system)  
*Set [SystemDynamics](#) object.*
- virtual int [dimension](#) () const  
*Return edge dimension by function call.*
- virtual [FUNCT\\_TYPE](#) [funcType](#) () const  
*Return function typed given by template parameter *FuncType*.*
- virtual const double \* [valuesData](#) () const  
*Return pointer to cost/constraint values [[dimension](#)() x 1] (read-only).*
- virtual double \* [valuesData](#) ()  
*Return pointer to cost/constraint values [[dimension](#)() x 1].*
- virtual const [ValueVector](#) & [values](#) () const  
*Return cost/constraint values as static Eigen::Vector (read-only).*
- virtual [ValueVector](#) & [values](#) ()  
*Return cost/constraint values as static Eigen::Vector.*
- virtual void [allocateMemory](#) (bool skip\_hessian=false)

- Allocate memory for JacobianWorkspace and HessianWorkspace and get vertices dimensions.*
- virtual [VertexType](#) \* [getVertex](#) (unsigned int idx)  
*Access attached vertex with index `idx`.*
  - virtual const [VertexType](#) \* [getVertex](#) (unsigned int idx) const  
*Access attached vertex with index `idx` (read-only).*
  - virtual unsigned int [noVertices](#) () const  
*Return the number of attached vertices.*
  - virtual void [calculateVertexDimensions](#) ()  
*Return the number of attached vertices.*
  - virtual const std::array  
< [VertexType](#) \*, [NoVertices](#) > & [vertices](#) () const  
*Return vertex container of the edge.*
  - virtual void [computeJacobian](#) ()  
*Actual cost function or constraint function. For constraints it is `computeValues()` <= 0.*
  - virtual void [computeHessian](#) ()  
*Compute Block-Hessian and store it to `EdgeType::_hessians` according to the description in `HessianWorkspace`.*
  - virtual const bool [isBoundConstraint](#) () const  
*Override this method in a subclass that constitutes a bound constraint since it can be handled separately by some solvers to speed up optimization.*
  - virtual [ValueVectorMap](#) [valuesMap](#) ()  
*Get cost/constraints values as Eigen type matrix.*
  - virtual void \* [getCustomData](#) ()  
*Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.*
  - virtual [JacobianWorkspace](#) & [jacobians](#) ()  
*Access the jacobian workspace of this edge.*
  - virtual const [JacobianWorkspace](#) & [jacobians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
  - virtual [HessianWorkspace](#) & [hessians](#) ()  
*Access the hessian workspace of this edge.*
  - virtual const [HessianWorkspace](#) & [hessians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
  - virtual void [backupJacobian](#) ()  
*Make a backup of the current jacobian block matrix.*
  - virtual void [restoreJacobian](#) ()  
*Restore jacobian block matrix from the backup stack.*
  - virtual void [discardJacobianBackup](#) ()  
*Discard current jacobian backup.*
  - virtual [JacobianWorkspace](#) & [getJacobianBackup](#) ()

### Static Public Attributes

- static const int [Dimension](#) = p  
*Dimension of this edge equals always the number of system equations.*
- static constexpr const int [NoVertices](#)

## Protected Attributes

- [SystemDynamics](#)< p, q > \* [\\_system](#) = nullptr  
*Pointer to the system dynamics object.*
- [ValueVector](#) [\\_values](#)  
*Actual cost/constraint value data object (initialized to zero).*
- const std::array< [VertexType](#) \*, [NoVertices](#) > [\\_vertices](#)  
*Container that stores all attached vertices.*
- std::array< int, [NoVertices](#) > [\\_vertices\\_dim](#)  
*Store dimensions for all vertices (EdgeType::\_vertices, [calculateVertexDimensions\(\)](#))*
- [JacobianWorkspace](#) [\\_jacobians](#)  
*Block-Jacobians of the edge (see [JacobianWorkspace](#)).*
- [HessianWorkspace](#) [\\_hessians](#)  
*Block-Hessians of the edge (see [HessianWorkspace](#)).*
- [BackupStackType](#)  
< [JacobianWorkspace](#) > [\\_jacob\\_backup](#)

## Friends

- template<int pf, int qf>  
class [SystemDynamics](#)

### 16.13.1 Detailed Description

```
template<int p, int q, bool central = false>class teb::EdgeSystemDynamics< p, q, central >
```

This edge creates an equality constraint based on nonlinear system-dynamics equations. System dynamics can be specified using a [SystemDynamics](#) object. The pointer to the system dynamics must be passed to the constructor of this object or by calling [setSystemObject\(\)](#). See [SystemDynamics](#) for more information about how to model a specific system.

This edge approximates the continuous-time derivatives that appear in each system equation by finite differences (see [teb::FiniteDifferences](#)). Two different types of finite differences are supported by this edge at the moment.

- Forward differences:  $\dot{x}(t) = \frac{x_{k+1} - x_k}{\Delta T}$
- Central differences:  $\dot{x}(t) = \frac{x_{k+1} - x_{k-1}}{2\Delta T}$

The number of vertices required by this edge depend on whether central differences are enabled or not:

- Forward differences:
  1. [StateVertex](#)  $\mathbf{x}_k$
  2. [ControlVertex](#)  $\mathbf{u}_k$
  3. [StateVertex](#)  $\mathbf{x}_{k+1}$
  4. [TimeDiff](#)  $\Delta T$
- Central differences:
  1. [StateVertex](#)  $\mathbf{x}_{k-1}$
  2. [StateVertex](#)  $\mathbf{x}_k$
  3. [ControlVertex](#)  $\mathbf{u}_k$
  4. [StateVertex](#)  $\mathbf{x}_{k+1}$
  1. [TimeDiff](#)  $\Delta T$



## Remarks

Call `allocateMemory()` before using the edge for optimization.

**Bug** Central differences do not seem to work as expected at the moment.

## Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

## Template Parameters

<i>p</i>	Number of state variables
<i>q</i>	Number of input variables
<i>central</i>	Enable central differences (make sure to read the info text -> vertex type may change)

Definition at line 52 of file base\_edge\_dynamics.h.

## 16.13.2 Member Typedef Documentation

16.13.2.1 `using teb::EdgeType::EdgeContainer = std::vector<EdgeType*> [inherited]`

Definition at line 114 of file graph.h.

16.13.2.2 `using teb::BaseEdge< D, FuncType, Vertices >::ValueVector = Eigen::Matrix<double, D, 1> [inherited]`

Definition at line 54 of file base\_edge.h.

16.13.2.3 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd > [inherited]`

Definition at line 116 of file graph.h.

## 16.13.3 Constructor &amp; Destructor Documentation

16.13.3.1 `template<int p, int q, bool central = false> teb::EdgeSystemDynamics< p, q, central >::EdgeSystemDynamics ( SystemDynamics< p, q > * system, StateVertex< p > & px1, ControlVertex< q > & pu1, StateVertex< p > & px2, TimeDiff & pdt ) [inline]`

## Parameters

<i>system</i>	Pointer to a <a href="#">SystemDynamics</a> object
<i>px1</i>	Pointer to a <a href="#">StateVertex</a> related to $\mathbf{x}_{k-1}$
<i>pu1</i>	Pointer to a <a href="#">ControlVertex</a> related to $\mathbf{u}_k$
<i>px2</i>	Pointer to a <a href="#">StateVertex</a> related to $\mathbf{x}_k$
<i>pdt</i>	Pointer to a <a href="#">TimeDiff</a> $\Delta T$

Definition at line 71 of file base\_edge\_dynamics.h.

## 16.13.4 Member Function Documentation

16.13.4.1 `virtual void teb::BaseEdge< D, FuncType, Vertices >::allocateMemory ( bool skip_hessian = false ) [inline], [virtual], [inherited]`

**Remarks**

Before calling, call `resizeVertexContainer()` edge and add all vertices (`setVertex()`) first.

**Todo** Implement automatic memory allocation instead calling this function manually.

**Parameters**

<i>skip_hessian</i>	If no hessian calculation is required by the solver skip memory allocation.
---------------------	---

Implements [teb::EdgeType](#).

Definition at line 84 of file `base_edge.h`.

**16.13.4.2** `virtual void teb::EdgeType::backupJacobian ( ) [inline],[virtual],[inherited]`

Definition at line 162 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.13.4.3** `virtual void teb::BaseEdge< D, FuncType, Vertices >::calculateVertexDimensions ( ) [inline],[virtual],[inherited]`

Query dimensions of all vertices attached to this edge and store them internally.

Stores dimensions to class container `EdgeType::_vertices_dim`. This function is called within `BaseEdge::allocateMemory()`.

**Todo** Maybe implement static/constexpr version of collecting all dimensions since they are known at compile-time (maybe <http://stackoverflow.com/questions/19019252/c11-create-0-to-n-constexpr-array->

Definition at line 110 of file `base_edge.h`.

**16.13.4.4** `virtual void teb::BaseEdge< D, FuncType, Vertices >::computeHessian ( ) [virtual],[inherited]`

This implementation calculates the Block-Hessians for all statically allocated vertices and cost/constraint values numerically using forward differences.

For more information about the Hessian structure refer to `HessianWorkspace`.

If the `FuncType` template parameter is set to `FUNCT_TYPE::LINEAR` or `FuncType == FUNCT_TYPE::NONLINEAR_ONCE_DIFF` the hessian is set to zero.

The results are stored to the class member `BaseEdgeBinary::_hessians`.

Feel free to reimplement this function in child classes in order to provide analytical Hessians.

**Remarks**

This function assumes that `computeJacobian()` was called before! It is used as reference for calculating the forward-differences.

**See Also**

[computeJacobian\(\)](#), `HessianWorkspace`

Implements [teb::EdgeType](#).

16.13.4.5 `virtual void teb::BaseEdge< D, FuncType, Vertices >::computeJacobian ( )` `[virtual]`, `[inherited]`

This implementation calculates the Block-Jacobians for all statically allocated vertices numerically using central differences.

Compute Block-Jacobian and store it to `EdgeType::_jacobians` according to the description in `JacobianWorkspace`.

For more information about the Jacobian structure refer to `JacobianWorkspace`.

The results are stored to the class member `BaseEdgeBinary::_jacobians`.

The implementation is similar to [2].

Feel free to reimplement this function in child classes in order to provide analytical Jacobians.

See Also

[computeHessian\(\)](#), `JacobianWorkspace`

Implements [teb::EdgeType](#).

16.13.4.6 `template<int p, int q, bool central = false> virtual void teb::EdgeSystemDynamics< p, q, central >::computeValues ( )` `[inline]`, `[virtual]`

Reimplemented from [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#).

Definition at line 76 of file `base_edge_dynamics.h`.

References `teb::EdgeSystemDynamics< p, q, central >::_system`, `teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >::_values`, `teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >::_vertices`, and `teb::StateVertex< p >::states()`.

16.13.4.7 `virtual int teb::BaseEdge< D, FuncType, Vertices >::dimension ( ) const` `[inline]`, `[virtual]`, `[inherited]`

Implements [teb::EdgeType](#).

Definition at line 51 of file `base_edge.h`.

16.13.4.8 `virtual void teb::EdgeType::discardJacobianBackup ( )` `[inline]`, `[virtual]`, `[inherited]`

Definition at line 171 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`.

16.13.4.9 `virtual FUNCT_TYPE teb::BaseEdge< D, FuncType, Vertices >::funcType ( ) const` `[inline]`, `[virtual]`, `[inherited]`

Implements [teb::EdgeType](#).

Definition at line 69 of file `base_edge.h`.

16.13.4.10 `virtual void* teb::EdgeType::getCustomData ( )` `[inline]`, `[virtual]`, `[inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 154 of file `graph.h`.

**16.13.4.11** `virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( ) [inline],[virtual],[inherited]`

Definition at line 173 of file graph.h.

References `teb::EdgeType::_jacob_backup`.

**16.13.4.12** `virtual VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 98 of file base\_edge.h.

**16.13.4.13** `virtual const VertexType* teb::BaseEdge< D, FuncType, Vertices >::getVertex ( unsigned int idx ) const [inline],[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 99 of file base\_edge.h.

**16.13.4.14** `virtual HessianWorkspace& teb::EdgeType::hessians ( ) [inline],[virtual],[inherited]`

Definition at line 159 of file graph.h.

References `teb::EdgeType::_hessians`.

**16.13.4.15** `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const [inline],[virtual],[inherited]`

Definition at line 160 of file graph.h.

References `teb::EdgeType::_hessians`.

**16.13.4.16** `virtual const bool teb::EdgeType::isBoundConstraint ( ) const [inline],[virtual],[inherited]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 124 of file graph.h.

**16.13.4.17** `virtual JacobianWorkspace& teb::EdgeType::jacobians ( ) [inline],[virtual],[inherited]`

Definition at line 157 of file graph.h.

References `teb::EdgeType::_jacobians`.

**16.13.4.18** `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const [inline],[virtual],[inherited]`

Definition at line 158 of file graph.h.

References `teb::EdgeType::_jacobians`.

**16.13.4.19** `virtual unsigned int teb::BaseEdge< D, FuncType, Vertices >::noVertices ( ) const [inline],  
[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 101 of file `base_edge.h`.

**16.13.4.20** `virtual void teb::EdgeType::restoreJacobian ( ) [inline],[virtual],[inherited]`

Definition at line 165 of file `graph.h`.

References `teb::EdgeType::_jacob_backup`, and `teb::EdgeType::_jacobians`.

**16.13.4.21** `template<int p, int q, bool central = false> void teb::EdgeSystemDynamics< p, q, central  
>::setSystemDynamics ( SystemDynamics< p, q > * system ) [inline]`

Parameters

<i>system</i>	Pointer to a <a href="#">SystemDynamics</a> object
---------------	--

Definition at line 93 of file `base_edge_dynamics.h`.

References `teb::EdgeSystemDynamics< p, q, central >::_system`.

**16.13.4.22** `virtual const ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) const [inline],  
[virtual],[inherited]`

Definition at line 74 of file `base_edge.h`.

**16.13.4.23** `virtual ValueVector& teb::BaseEdge< D, FuncType, Vertices >::values ( ) [inline],[virtual],  
[inherited]`

Definition at line 75 of file `base_edge.h`.

**16.13.4.24** `virtual const double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) const [inline],  
[virtual],[inherited]`

Implements [teb::EdgeType](#).

Definition at line 72 of file `base_edge.h`.

**16.13.4.25** `virtual double* teb::BaseEdge< D, FuncType, Vertices >::valuesData ( ) [inline],[virtual],  
[inherited]`

Implements [teb::EdgeType](#).

Definition at line 73 of file `base_edge.h`.

**16.13.4.26** `virtual ValueVectorMap teb::EdgeType::valuesMap ( ) [inline],[virtual],[inherited]`

return `Eigen::Vector` type that maps to the actual cost/constraints values

Definition at line 150 of file `graph.h`.

References `teb::EdgeType::dimension()`, and `teb::EdgeType::valuesData()`.

**16.13.4.27** `virtual const std::array<VertexType*,NoVertices>& teb::BaseEdge< D, FuncType, Vertices >::vertices ( )`  
`const [inline],[virtual],[inherited]`

#### Returns

(Read-only) reference to the vertex container.

Definition at line 122 of file base\_edge.h.

### 16.13.5 Friends And Related Function Documentation

**16.13.5.1** `template<int p, int q, bool central = false> template<int pf, int qf> friend class SystemDynamics [friend]`

Definition at line 55 of file base\_edge\_dynamics.h.

### 16.13.6 Member Data Documentation

**16.13.6.1** `HessianWorkspace teb::EdgeType::_hessians [protected],[inherited]`

Definition at line 178 of file graph.h.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::EdgeMinimizeTime::computeHessian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian()`, `teb::EdgeQuadraticForm< p, q >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, and `teb::EdgeType::hessians()`.

**16.13.6.2** `BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup [protected],[inherited]`

Definition at line 180 of file graph.h.

Referenced by `teb::EdgeType::backupJacobian()`, `teb::EdgeType::discardJacobianBackup()`, `teb::EdgeType::getJacobianBackup()`, and `teb::EdgeType::restoreJacobian()`.

**16.13.6.3** `JacobianWorkspace teb::EdgeType::_jacobians [protected],[inherited]`

Definition at line 173 of file graph.h.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::EdgeType::backupJacobian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::EdgeMinimizeTime::computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian()`, `teb::EdgeQuadraticForm< p, q >::computeJacobian()`, `teb::BaseEdge< D, FuncType >::computeJacobian()`, `teb::EdgeType::jacobians()`, and `teb::EdgeType::restoreJacobian()`.

**16.13.6.4** `template<int p, int q, bool central = false> SystemDynamics<p, q>* teb::EdgeSystemDynamics< p, q, central >::_system = nullptr [protected]`

Definition at line 102 of file base\_edge\_dynamics.h.

Referenced by `teb::EdgeSystemDynamics< p, q, central >::computeValues()`, and `teb::EdgeSystemDynamics< p, q, central >::setSystemDynamics()`.

**16.13.6.5** `ValueVector teb::BaseEdge< D, FuncType, Vertices >::_values [protected],[inherited]`

Definition at line 139 of file base\_edge.h.

Referenced by `teb::EdgeSystemDynamics< p, q, central >::computeValues()`.

16.13.6.6 `const std::array<VertexType*, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices`  
`[protected], [inherited]`

Definition at line 141 of file `base_edge.h`.

Referenced by `teb::EdgeSystemDynamics< p, q, central >::computeValues()`.

16.13.6.7 `std::array<int, NoVertices> teb::BaseEdge< D, FuncType, Vertices >::_vertices_dim` `[protected], [inherited]`

Definition at line 142 of file `base_edge.h`.

16.13.6.8 `template<int p, int q, bool central = false> const int teb::EdgeSystemDynamics< p, q, central >::Dimension = p` `[static]`

Definition at line 59 of file `base_edge_dynamics.h`.

16.13.6.9 `constexpr const int teb::BaseEdge< D, FuncType, Vertices >::NoVertices` `[static], [inherited]`

Definition at line 48 of file `base_edge.h`.

The documentation for this class was generated from the following file:

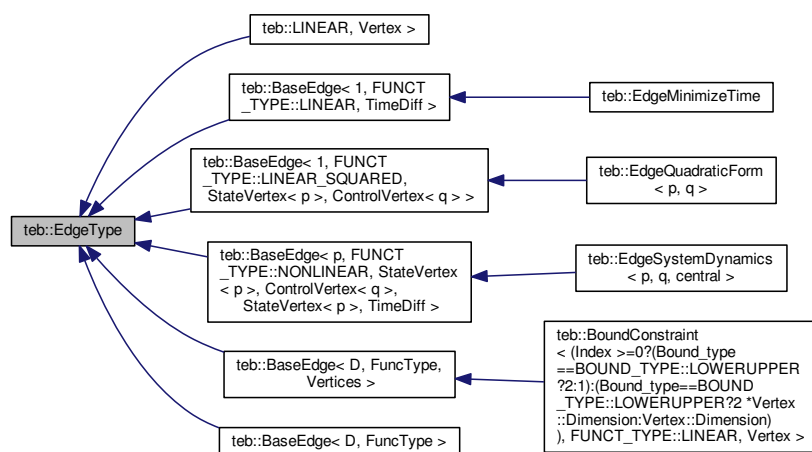
- [base\\_edge\\_dynamics.h](#)

## 16.14 teb::EdgeType Class Reference

Generic interface class for edges.

```
#include <graph.h>
```

Inheritance diagram for `teb::EdgeType`:



### Public Types

- using [EdgeContainer](#) = `std::vector< EdgeType * >`

*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

- using [ValueVectorMap](#) = Eigen::Map< Eigen::VectorXd >

*Typedef for an Eigen::Vector wrapper that points to the actual data somewhere else.*

## Public Member Functions

- virtual [FUNCT\\_TYPE](#) [funcType](#) () const =0  
*Store edge function type information according to the members of enum FUNCT\_TYPE.*
- virtual const bool [isBoundConstraint](#) () const  
*Override this method in a subclass that constitutes a bound constraint since it can be handled seperately by some solvers to speed up optimization.*
- [EdgeType](#) ()  
*Constructor that updates the internal edge\_type.*
- virtual [~EdgeType](#) ()
- virtual int [dimension](#) () const =0  
*Empty destructor.*
- virtual void [allocateMemory](#) (bool skip\_hessian=false)=0  
*Allocate memory to store cost/constraint values, jacobians and hessians.*
- virtual void [computeValues](#) ()=0  
*Actual cost function or constraint function. For constraints it is [computeValues\(\)](#) <= 0.*
- virtual void [computeJacobian](#) ()=0  
*Compute Block-Jacobian and store it to [EdgeType::\\_jacobians](#) according to the description in [JacobianWorkspace](#).*
- virtual void [computeHessian](#) ()=0  
*Compute Block-Hessian and store it to [EdgeType::\\_hessians](#) according to the description in [HessianWorkspace](#).*
- virtual const double \* [valuesData](#) () const =0  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1] (read-only).*
- virtual double \* [valuesData](#) ()=0  
*Return pointer to cost/constraint values [[dimension\(\)](#) x 1].*
- virtual unsigned int [noVertices](#) () const =0  
*Return the number of attached vertices.*
- virtual [VertexType](#) \* [getVertex](#) (unsigned int idx)=0  
*Access attached vertex with index [idx](#).*
- virtual const [VertexType](#) \* [getVertex](#) (unsigned int idx) const =0  
*Access attached vertex with index [idx](#) (read-only).*
- virtual [ValueVectorMap](#) [valuesMap](#) ()  
*Get cost/constraints values as Eigen type matrix.*
- virtual void \* [getCustomData](#) ()  
*Use this function to return a pointer to custom data that can be used by the solver without knowing template parameters e.g.*
- virtual [JacobianWorkspace](#) & [jacobians](#) ()  
*Access the jacobian workspace of this edge.*
- virtual const [JacobianWorkspace](#) & [jacobians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual [HessianWorkspace](#) & [hessians](#) ()  
*Access the hessian workspace of this edge.*
- virtual const [HessianWorkspace](#) & [hessians](#) () const  
*Access the jacobian workspace of this edge (read-only).*
- virtual void [backupJacobian](#) ()  
*Make a backup of the current jacobian block matrix.*
- virtual void [restoreJacobian](#) ()  
*Restore jacobian block matrix from the backup stack.*
- virtual void [discardJacobianBackup](#) ()  
*Discard current jacobian backup.*
- virtual [JacobianWorkspace](#) & [getJacobianBackup](#) ()



## Protected Attributes

- [JacobianWorkspace \\_jacobians](#)  
*Block-Jacobians of the edge (see [JacobianWorkspace](#)).*
- [HessianWorkspace \\_hessians](#)  
*Block-Hessians of the edge (see [HessianWorkspace](#)).*
- [BackupStackType](#)  
< [JacobianWorkspace](#) > `_jacob_backup`

### 16.14.1 Detailed Description

This abstract class defines the interface for dedicated edges. The underlying TEB optimization problem is formulated as a hyper-graph in which state vectors ([StateVertex](#)), control input vectors ([ControlVertex](#)) and a [TimeDiff](#) are represented as vertices. Cost functions and constraints (that depend on vertices) are formulated as multi-edges. Multi-edges can connect an arbitrary number of vertices. The result graph is therefore called hyper-graph. If the number of vertices attached to a single edge is low, the resulting optimization problem will be sparse probably. Block-Jacobians and Block-Hessians can be calculated for each edge independently and may be combined afterwards by the solver.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### See Also

[HyperGraph](#), [VertexType](#), [BaseEdge](#)

Definition at line 109 of file `graph.h`.

### 16.14.2 Member Typedef Documentation

16.14.2.1 `using teb::EdgeType::EdgeContainer = std::vector<EdgeType*>`

Definition at line 114 of file `graph.h`.

16.14.2.2 `using teb::EdgeType::ValueVectorMap = Eigen::Map< Eigen::VectorXd >`

Definition at line 116 of file `graph.h`.

### 16.14.3 Constructor & Destructor Documentation

16.14.3.1 `teb::EdgeType::EdgeType ( )` `[inline]`

Definition at line 127 of file `graph.h`.

16.14.3.2 `virtual teb::EdgeType::~~EdgeType ( )` `[inline]`, `[virtual]`

Definition at line 131 of file `graph.h`.

### 16.14.4 Member Function Documentation

16.14.4.1 `virtual void teb::EdgeType::allocateMemory ( bool skip_hessian = false ) [pure virtual]`

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), and [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

16.14.4.2 `virtual void teb::EdgeType::backupJacobian ( ) [inline],[virtual]`

Definition at line 162 of file graph.h.

References [\\_jacob\\_backup](#), and [\\_jacobians](#).

16.14.4.3 `virtual void teb::EdgeType::computeHessian ( ) [pure virtual]`

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::EdgeQuadraticForm< p, q >](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#), and [teb::EdgeMinimizeTime](#).

16.14.4.4 `virtual void teb::EdgeType::computeJacobian ( ) [pure virtual]`

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::EdgeQuadraticForm< p, q >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#), and [teb::EdgeMinimizeTime](#).

16.14.4.5 `virtual void teb::EdgeType::computeValues ( ) [pure virtual]`

Implemented in [teb::EdgeQuadraticForm< p, q >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#), [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#), [teb::EdgeSystemDynamics< p, q, central >](#), and [teb::EdgeMinimizeTime](#).

16.14.4.6 `virtual int teb::EdgeType::dimension ( ) const [pure virtual]`

Return number of scalar cost/constraint values covered by this edge.

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), and [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

Referenced by [valuesMap\(\)](#).

16.14.4.7 `virtual void teb::EdgeType::discardJacobianBackup ( ) [inline],[virtual]`

Definition at line 171 of file graph.h.

References [\\_jacob\\_backup](#).

16.14.4.8 `virtual FUNCT_TYPE teb::EdgeType::funcType ( ) const` `[pure virtual]`

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), and [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

16.14.4.9 `virtual void* teb::EdgeType::getCustomData ( )` `[inline],[virtual]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 154 of file `graph.h`.

16.14.4.10 `virtual JacobianWorkspace& teb::EdgeType::getJacobianBackup ( )` `[inline],[virtual]`

Definition at line 173 of file `graph.h`.

References `_jacob_backup`.

16.14.4.11 `virtual VertexType* teb::EdgeType::getVertex ( unsigned int idx )` `[pure virtual]`

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), and [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

16.14.4.12 `virtual const VertexType* teb::EdgeType::getVertex ( unsigned int idx ) const` `[pure virtual]`

Implemented in [teb::BaseEdge< D, FuncType >](#), [teb::BaseEdge< D, FuncType, Vertices >](#), [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR\\_SQUARED, StateVertex< p >, ControlVertex< q > >](#), [teb::BaseEdge< p, FUNCT\\_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >](#), [teb::LINEAR, Vertex >](#), and [teb::BaseEdge< 1, FUNCT\\_TYPE::LINEAR, TimeDiff >](#).

16.14.4.13 `virtual HessianWorkspace& teb::EdgeType::hessians ( )` `[inline],[virtual]`

Definition at line 159 of file `graph.h`.

References `_hessians`.

16.14.4.14 `virtual const HessianWorkspace& teb::EdgeType::hessians ( ) const` `[inline],[virtual]`

Definition at line 160 of file `graph.h`.

References `_hessians`.

16.14.4.15 `virtual const bool teb::EdgeType::isBoundConstraint ( ) const` `[inline],[virtual]`

Reimplemented in [teb::BoundConstraint< Bound\\_type, Vertex, Bound\\_vars, Index >](#).

Definition at line 124 of file `graph.h`.

16.14.4.16 `virtual JacobianWorkspace& teb::EdgeType::jacobians ( )` `[inline],[virtual]`

Definition at line 157 of file `graph.h`.

References `_jacobians`.

**16.14.4.17** `virtual const JacobianWorkspace& teb::EdgeType::jacobians ( ) const` `[inline],[virtual]`

Definition at line 158 of file `graph.h`.

References `_jacobians`.

**16.14.4.18** `virtual unsigned int teb::EdgeType::noVertices ( ) const` `[pure virtual]`

Implemented in `teb::BaseEdge< D, FuncType >`, `teb::BaseEdge< D, FuncType, Vertices >`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >`, `teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >`, `teb::LINEAR, Vertex >`, and `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >`.

**16.14.4.19** `virtual void teb::EdgeType::restoreJacobian ( )` `[inline],[virtual]`

Definition at line 165 of file `graph.h`.

References `_jacob_backup`, and `_jacobians`.

**16.14.4.20** `virtual const double* teb::EdgeType::valuesData ( ) const` `[pure virtual]`

Implemented in `teb::BaseEdge< D, FuncType >`, `teb::BaseEdge< D, FuncType, Vertices >`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >`, `teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >`, `teb::LINEAR, Vertex >`, and `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >`.

Referenced by `valuesMap()`.

**16.14.4.21** `virtual double* teb::EdgeType::valuesData ( )` `[pure virtual]`

Implemented in `teb::BaseEdge< D, FuncType >`, `teb::BaseEdge< D, FuncType, Vertices >`, `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR_SQUARED, StateVertex< p >, ControlVertex< q > >`, `teb::BaseEdge< p, FUNCT_TYPE::NONLINEAR, StateVertex< p >, ControlVertex< q >, StateVertex< p >, TimeDiff >`, `teb::LINEAR, Vertex >`, and `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >`.

**16.14.4.22** `virtual ValueVectorMap teb::EdgeType::valuesMap ( )` `[inline],[virtual]`

return `Eigen::Vector` type that maps to the actual cost/constraints values

Definition at line 150 of file `graph.h`.

References `dimension()`, and `valuesData()`.

## 16.14.5 Member Data Documentation

**16.14.5.1** `HessianWorkspace` `teb::EdgeType::_hessians` `[protected]`

Definition at line 178 of file `graph.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `teb::EdgeMinimizeTime::computeHessian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian()`, `teb::EdgeQuadraticForm< p, q >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, and `hessians()`.

16.14.5.2 `BackupStackType<JacobianWorkspace> teb::EdgeType::_jacob_backup` `[protected]`

Definition at line 180 of file `graph.h`.

Referenced by `backupJacobian()`, `discardJacobianBackup()`, `getJacobianBackup()`, and `restoreJacobian()`.

16.14.5.3 `JacobianWorkspace teb::EdgeType::_jacobians` `[protected]`

Definition at line 173 of file `graph.h`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, `teb::BaseEdge< D, FuncType >::allocateMemory()`, `backupJacobian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::EdgeMinimizeTime::computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian()`, `teb::EdgeQuadraticForm< p, q >::computeJacobian()`, `teb::BaseEdge< D, FuncType >::computeJacobian()`, `jacobians()`, and `restoreJacobian()`.

The documentation for this class was generated from the following file:

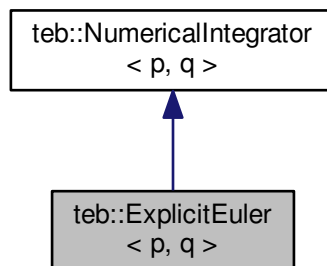
- [graph.h](#)

16.15 `teb::ExplicitEuler< p, q >` Class Template Reference

Simple explicit euler method for integration.

```
#include <integrators.h>
```

Inheritance diagram for `teb::ExplicitEuler< p, q >`:



## Public Types

- using `StateVector` = typename `NumericalIntegrator< p, q >::StateVector`  
*Typedef for state vector with p states [p x 1].*
- using `ControlVector` = typename `NumericalIntegrator< p, q >::ControlVector`  
*Typedef for control input vector with q controls [q x 1].*

## Public Member Functions

- `ExplicitEuler()`  
*Empty Constructor.*

- virtual `~ExplicitEuler()`  
*Empty Destructor.*
- virtual `StateVector integrate` (const Eigen::Ref< const `StateVector` > &x\_k, const Eigen::Ref< const `ControlVector` > &u\_k, `SystemDynamics`< p, q > \*system\_equation, double dt) const

*Integrate a nonlinear state-space model*

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

### 16.15.1 Detailed Description

```
template<int p, int q>class teb::ExplicitEuler< p, q >
```

For more information please refer to [http://en.wikipedia.org/wiki/Euler\\_method](http://en.wikipedia.org/wiki/Euler_method)

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### Template Parameters

<i>p</i>	Number of state variables
<i>q</i>	Number of input variables

Definition at line 81 of file integrators.h.

### 16.15.2 Member Typedef Documentation

16.15.2.1 `template<int p, int q> using teb::ExplicitEuler< p, q >::ControlVector = typename NumericalIntegrator<p,q>::ControlVector`

Definition at line 87 of file integrators.h.

16.15.2.2 `template<int p, int q> using teb::ExplicitEuler< p, q >::StateVector = typename NumericalIntegrator<p,q>::StateVector`

Definition at line 85 of file integrators.h.

### 16.15.3 Constructor & Destructor Documentation

16.15.3.1 `template<int p, int q> teb::ExplicitEuler< p, q >::ExplicitEuler ( ) [inline]`

Definition at line 89 of file integrators.h.

16.15.3.2 `template<int p, int q> virtual teb::ExplicitEuler< p, q >::~~ExplicitEuler ( ) [inline], [virtual]`

Definition at line 90 of file integrators.h.

### 16.15.4 Member Function Documentation

16.15.4.1 `template<int p, int q> virtual StateVector teb::ExplicitEuler< p, q >::integrate ( const Eigen::Ref< const StateVector > & x_k, const Eigen::Ref< const ControlVector > & u_k, SystemDynamics< p, q > * system_equation, double dt ) const [inline], [virtual]`

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t.  $\mathbf{x}$ . In particular, it relies on discrete states. The method calculates  $\mathbf{x}_{k+1}$  based on  $\mathbf{x}_k$ ,  $\mathbf{u}_k$  and the duration  $\Delta T$ . The dedicated system dynamics equations may be specified using a [SystemDynamics](#) object.

Parameters

$x_k$	Current state vector $\mathbf{x}_k$ [p x 1]
$u_k$	Current state vector $\mathbf{u}_k$ [q x 1]
<i>system_equation</i>	Pointer to the <a href="#">SystemDynamics</a> object that stores the system equations.
<i>dt</i>	Time interval for the integration

Returns

Integrated state vector  $\mathbf{x}_{k+1}$  [p x 1]

Implements [teb::NumericalIntegrator< p, q >](#).

Definition at line 93 of file `integrators.h`.

References [teb::SystemDynamics< p, q >::stateSpaceModel\(\)](#).

The documentation for this class was generated from the following file:

- [integrators.h](#)

## 16.16 teb::Config::Optim::Solver::NonlinearProgram::Hessian Struct Reference

Configurations for the hessian of the lagrangian calculation.

```
#include <config.h>
```

Public Attributes

- [HessianMethod hessian\\_method = HessianMethod::NUMERIC](#)  
*Define the hessian calculation methods (numeric, bfgs quasi-newton, ...)*
- [HessianInit hessian\\_init = HessianInit::NUMERIC](#)  
*Define which hessian initialization should be used in BFGS methods.*
- double [hessian\\_init\\_identity\\_scale = 1](#)  
*Scale the hessian initialization matrix if hessian\_init is set to [HessianInit::IDENTITY](#).*
- bool [bfgs\\_damped\\_mode = true](#)  
*Specify if BFGS damped mode should be used (slower, but more robust).*

### 16.16.1 Detailed Description

Definition at line 96 of file `config.h`.

## 16.16.2 Member Data Documentation

16.16.2.1 `bool teb::Config::Optim::Solver::NonlinearProgram::Hessian::bfgs_damped_mode = true`

Definition at line 101 of file `config.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`.

16.16.2.2 `HessianInit teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init = HessianInit::NUMERIC`

Definition at line 99 of file `config.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`.

16.16.2.3 `double teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init_identity_scale = 1`

Definition at line 100 of file `config.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`.

16.16.2.4 `HessianMethod teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method = HessianMethod::NUMERIC`

Definition at line 98 of file `config.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

The documentation for this struct was generated from the following file:

- [config.h](#)

## 16.17 teb::HessianWorkspace Class Reference

Memory management and accessor functions for the Block-Hessian (for each edge).

```
#include <workspaces.h>
```

### Public Types

- using [WorkspaceMatrix](#) = `std::vector< std::vector< Eigen::MatrixXd, Eigen::aligned_allocator< Eigen::MatrixXd >>>`

*Typedef for the BlockHessian container.*

### Public Member Functions

- [HessianWorkspace](#) ()
- [~HessianWorkspace](#) ()  
*Empty constructor.*
- `template<typename T >`  
`bool allocate (int edge_dim, const T &vertices_dim)`  
*Empty destructor.*
- `Eigen::MatrixXd & getWorkspace (int vertex_idx, int value_idx)`  
*Access Block-Hessian for TebVertex with index `vertex_idx`.*



## Protected Attributes

- [WorkspaceMatrix \\_workspace](#)

*Block-Hessian container.*

### 16.17.1 Detailed Description

This class controls the memory management of Block-Hessians that are calculated for each edge (depends on the [BaseSolver](#) subclass, if Hessians are required). Each edge instantiates its own [HessianWorkspace](#) and allocates the memory required to store all entries. Since the size depends on the type and number of attached vertices, it is not known a-priori. This workspace helps managing memory and accessing elements at run-time.

#### Remarks

For each vertex attached to a subclass of [BaseEdge](#) and addition for each cost-value (!), a single Hessian of the size [ [TebVertex::dimension\(\)](#) x [TebVertex::dimension\(\)](#) ] is required.

Assume that there exist a vertex1 with the following unfixed variables: [x1, x2, x3]. In addition let vertex2 be a vertex with the unfixed variables: [x4,x5]. For the an edge with two cost/constraint values [v1, v2], the corresponding Block-Hessians are as follow:

$$\mathbf{H}_{block1,vertex1} = \begin{bmatrix} \partial_{x1,x1}^2 v1 & \partial_{x1,x2}^2 v1 & \partial_{x1,x3}^2 v1 \\ \partial_{x2,x1}^2 v1 & \partial_{x2,x2}^2 v1 & \partial_{x2,x3}^2 v1 \\ \partial_{x3,x1}^2 v1 & \partial_{x3,x2}^2 v1 & \partial_{x3,x3}^2 v1 \end{bmatrix}$$

$$\mathbf{H}_{block1,vertex2} = \begin{bmatrix} \partial_{x4,x4}^2 v1 & \partial_{x4,x5}^2 v1 \\ \partial_{x5,x4}^2 v1 & \partial_{x5,x5}^2 v1 \end{bmatrix}$$

$$\mathbf{H}_{block2,vertex1} = \begin{bmatrix} \partial_{x1,x1}^2 v2 & \partial_{x1,x2}^2 v2 & \partial_{x1,x3}^2 v2 \\ \partial_{x2,x1}^2 v2 & \partial_{x2,x2}^2 v2 & \partial_{x2,x3}^2 v2 \\ \partial_{x3,x1}^2 v2 & \partial_{x3,x2}^2 v2 & \partial_{x3,x3}^2 v2 \end{bmatrix}$$

$$\mathbf{H}_{block2,vertex2} = \begin{bmatrix} \partial_{x4,x4}^2 v2 & \partial_{x4,x5}^2 v2 \\ \partial_{x5,x4}^2 v2 & \partial_{x5,x5}^2 v2 \end{bmatrix}$$

The above Block-Hessians can be stored from the edge class (after calling [allocate\(\)](#) once) as follows:

```
this->getWorkspace(0,0) = H_block1_vertex1; // exchange "this" by a pointer/ref to a valid
workspace instance with allocated and reserved memory.
this->getWorkspace(1,0) = H_block1_vertex2;
this->getWorkspace(1,1) = H_block2_vertex1;
this->getWorkspace(1,1) = H_block2_vertex2;
```

**Todo** [TebVertex::dimension\(\)](#) is used at the moment. Acutally only a number of [TebVertex::dimensionFree\(\)](#) non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

[BaseEdge](#), [JacobianWorkspace](#)

Definition at line 153 of file workspaces.h.

## 16.17.2 Member Typedef Documentation

16.17.2.1 `using teb::HessianWorkspace::WorkspaceMatrix = std::vector<std::vector<Eigen::MatrixXd, Eigen::aligned_allocator<Eigen::MatrixXd>>>`

A single block-matrix per attached TebVertex and per Cost-value is required.

Definition at line 157 of file workspaces.h.

## 16.17.3 Constructor & Destructor Documentation

16.17.3.1 `teb::HessianWorkspace::HessianWorkspace ( ) [inline]`

Definition at line 159 of file workspaces.h.

16.17.3.2 `teb::HessianWorkspace::~~HessianWorkspace ( ) [inline]`

Definition at line 160 of file workspaces.h.

## 16.17.4 Member Function Documentation

16.17.4.1 `template<typename T> bool teb::HessianWorkspace::allocate ( int edge_dim, const T & vertices_dim ) [inline]`

Allocate memory for all block-Jacobians.

Parameters

<i>edge_dim</i>	Dimension of the edge. Typically <a href="#">BaseEdge::Dimension()</a> .
<i>vertices_dim</i>	Vector of dimensions of the attached vertices. Typically <a href="#">TebVertex::dimension()</a> each.

Template Parameters

<i>T</i>	storage class (std::vector, std::array, std::deque)
----------	---

Return values

<i>true</i>	Status flag. Currently it returns only <code>true</code> .
-------------	--

Definition at line 170 of file workspaces.h.

References `_workspace`.

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, and `teb::BaseEdge< D, FuncType >::allocateMemory()`.

16.17.4.2 `Eigen::MatrixXd& teb::HessianWorkspace::getWorkspace ( int vertex_idx, int value_idx ) [inline]`

Parameters

<i>vertex_idx</i>	Index of the corresponding vertex.
<i>value_idx</i>	Index of the corresponding cost-value.

Returns

Reference to Block-Jacobian of vertex with index `vertex_idx` and cost value with index `value_idx`.

Definition at line 188 of file workspaces.h.

References `_workspace`.

Referenced by `teb::EdgeMinimizeTime::computeHessian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeHessian()`, `teb::EdgeQuadraticForm< p, q >::computeHessian()`, and `teb::BaseEdge< D, FuncType >::computeHessian()`.

## 16.17.5 Member Data Documentation

### 16.17.5.1 WorkspaceMatrix `teb::HessianWorkspace::_workspace` `[protected]`

Definition at line 188 of file `workspaces.h`.

Referenced by `allocate()`, and `getWorkspace()`.

The documentation for this class was generated from the following file:

- [workspaces.h](#)

## 16.18 teb::HyperGraph Class Reference

Graph object that stores pointers to active vertices and edges.

```
#include <graph.h>
```

### Public Types

- using `EdgeContainer` = `EdgeType::EdgeContainer`
- using `VertexContainer` = `VertexType::VertexContainer`

### Public Member Functions

- `EdgeContainer & objectives ()`  
*Access objective container.*
- `const EdgeContainer & objectives () const`  
*Access all edges containing objectives (read-only)*
- `EdgeContainer & equalities ()`  
*Access equality constraint container.*
- `const EdgeContainer & equalities () const`  
*Access all edges containing inequality constraints (read-only)*
- `EdgeContainer & inequalities ()`  
*Access inequality constraint container.*
- `const EdgeContainer & inequalities () const`  
*Access all edges containing equality constraints (read-only)*
- `VertexContainer & activeVertices ()`  
*Access container that stores only the active vertices of the hyper-graph.*
- `const VertexContainer & activeVertices () const`  
*Access all **active** vertices (read-only)*
- `void clearEdges ()`  
*Clears all existing edges and frees memory.*
- `void clearActiveVertices ()`  
*Clears all active vertices (the actual vertices remain untouched).*
- `void clearGraph ()`  
*Clear the complete graph.*

- void [addActiveVertex](#) ([VertexType](#) \*vertex)  
*Add new vertex to the graph.*
- void [addEdgeObjective](#) ([EdgeType](#) \*edge)  
*Add new objective edge to the graph.*
- void [addEdgeInequality](#) ([EdgeType](#) \*edge)  
*Add new inequality constraint edge to the graph.*
- void [addEdgeEquality](#) ([EdgeType](#) \*edge)  
*Add new equality constraint edge to the graph.*
- void [notifyGraphModified](#) (bool modified=true)  
*Set status of the graph to modified.*
- bool [isGraphModified](#) () const  
*Return status that tracks if the graph has been modified The status is tracked in order to allow the controller class and the solver to hotstart from previous settings.*

### Protected Attributes

- [EdgeContainer \\_objectives](#)  
*Store all edges containing local objective/cost functions (see [buildOptimizationGraph\(\)](#)).*
- [EdgeContainer \\_constraints\\_eq](#)  
*Store all edges containing equality constraints (see [buildOptimizationGraph\(\)](#)).*
- [EdgeContainer \\_constraints\\_ineq](#)  
*Store all edges containing inequality constraints (see [buildOptimizationGraph\(\)](#)).*
- [VertexContainer \\_active\\_vertices](#)  
*Store all active vertices (active = at least one unfixed variable, see [getActiveVertices\(\)](#)).*
- bool [\\_graph\\_modified](#) = true  
*Store status flag if the graph structure is changed or not.*

### 16.18.1 Detailed Description

This class stores pointer to all active vertices and edges in the optimizable hyper-graph. An instance of this class needs to be created inside the controller class and afterwards it is passed to a dedicated solver class via [BaseSolver::solve\(\)](#). The solver solves the optimization problem based on the hyper-graph specified within this class.

Note: Active vertices are all vertices with at least one degree of freedom.

#### Remarks

The graph takes ownership for the memory management of the edges. Vertices remain untouched, since they are pointing to real states in the controller class (and only to the active ones, only the vector of pointers will be cleared).

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### See Also

[VertexType](#), [EdgeType](#), [StateVertex](#), [ControlVertex](#), [TimeDiff](#), [BaseEdge](#)

Definition at line 205 of file [graph.h](#).

## 16.18.2 Member Typedef Documentation

### 16.18.2.1 using teb::HyperGraph::EdgeContainer = EdgeType::EdgeContainer

Definition at line 208 of file graph.h.

### 16.18.2.2 using teb::HyperGraph::VertexContainer = VertexType::VertexContainer

Definition at line 209 of file graph.h.

## 16.18.3 Member Function Documentation

### 16.18.3.1 VertexContainer& teb::HyperGraph::activeVertices ( ) [inline]

Active vertices are vertices that store at least one free optimization variable. That means, that iff a vertex (e.g. [StateVertex](#), [ControlVertex](#) or [TimeDiff](#)) is completely fixed, than it is not active. In that case the vertex could be skipped during optimization to speed up optimization time.

#### Returns

Reference to the active vertices container.

Definition at line 264 of file graph.h.

References `_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

### 16.18.3.2 const VertexContainer& teb::HyperGraph::activeVertices ( ) const [inline]

Definition at line 266 of file graph.h.

References `_active_vertices`.

### 16.18.3.3 void teb::HyperGraph::addActiveVertex ( VertexType \* vertex ) [inline]

The graph does not take over memory management for vertices.

#### Parameters

<i>vertex</i>	Pointer to the <a href="#">VertexType</a> subclass.
---------------	---

Definition at line 312 of file graph.h.

References `_active_vertices`.

### 16.18.3.4 void teb::HyperGraph::addEdgeEquality ( EdgeType \* edge ) [inline]

The graph takes care about memory management for this edge.

#### Parameters

<i>edge</i>	Pointer to the <a href="#">EdgeType</a> subclass.
-------------	---

Definition at line 344 of file graph.h.

References `_constraints_eq`.

16.18.3.5 `void teb::HyperGraph::addEdgeInequality ( EdgeType * edge )` `[inline]`

The graph takes care about memory management for this edge.

## Parameters

<i>edge</i>	Pointer to the <a href="#">EdgeType</a> subclass.
-------------	---

Definition at line 333 of file graph.h.

References `_constraints_ineq`.

**16.18.3.6** `void teb::HyperGraph::addEdgeObjective ( EdgeType * edge ) [inline]`

The graph takes care about memory management for this edge.

## Parameters

<i>edge</i>	Pointer to the <a href="#">EdgeType</a> subclass.
-------------	---

Definition at line 322 of file graph.h.

References `_objectives`.

**16.18.3.7** `void teb::HyperGraph::clearActiveVertices ( ) [inline]`

Definition at line 288 of file graph.h.

References `_active_vertices`, and `_graph_modified`.

Referenced by `clearGraph()`.

**16.18.3.8** `void teb::HyperGraph::clearEdges ( ) [inline]`

Clears all edges (objectives, inequality constraints and equality constraints). All memory will be freed.

Definition at line 274 of file graph.h.

References `_constraints_eq`, `_constraints_ineq`, `_graph_modified`, and `_objectives`.

Referenced by `clearGraph()`.

**16.18.3.9** `void teb::HyperGraph::clearGraph ( ) [inline]`

Clears all edges (objectives, inequality constraints and equality constraints). All memory for the edges will be freed. The actual vertices remain untouched.

Definition at line 300 of file graph.h.

References `clearActiveVertices()`, and `clearEdges()`.

Referenced by `teb::TebController< p, q >::resetController()`, and `teb::TebController< p, q >::~~TebController()`.

**16.18.3.10** `EdgeContainer& teb::HyperGraph::equalities ( ) [inline]`

Equality constraints (subclasses of [BaseEdge](#)) are inserted within [TebController::buildOptimizationGraph\(\)](#) and [TebController::customOptimizationGraph\(\)](#).

## Returns

Reference to the equality constraint container.

Definition at line 240 of file graph.h.

References `_constraints_eq`.

Referenced by `teb::BaseSolver::solve()`.

**16.18.3.11** `const EdgeContainer& teb::HyperGraph::equalities ( ) const` `[inline]`

Definition at line 242 of file graph.h.

References `_constraints_eq`.

**16.18.3.12** `EdgeContainer& teb::HyperGraph::inequalities ( )` `[inline]`

Inequality constraints (subclasses of [BaseEdge](#)) are inserted within [TebController::buildOptimizationGraph\(\)](#) and [TebController::customOptimizationGraph\(\)](#).

#### Returns

Reference to the inequality constraint container.

Definition at line 251 of file graph.h.

References `_constraints_ineq`.

Referenced by `teb::BaseSolver::solve()`.

**16.18.3.13** `const EdgeContainer& teb::HyperGraph::inequalities ( ) const` `[inline]`

Definition at line 253 of file graph.h.

References `_constraints_ineq`.

**16.18.3.14** `bool teb::HyperGraph::isGraphModified ( ) const` `[inline]`

If the structure of the optimization problem remains unchanged, one do not need to collect all active edges again. Therefore check the status using this function whenever vertices and edges are processed.

#### Returns

`true` if the graph has been modified

Definition at line 373 of file graph.h.

References `_graph_modified`.

Referenced by `teb::BaseSolver::solve()`.

**16.18.3.15** `void teb::HyperGraph::notifyGraphModified ( bool modified = true )` `[inline]`

The status is tracked in order to allow the controller class and the solver to hotstart from previous settings. If the structure of the optimization problem remains unchanged, one do not need to collect all active edges again. Therefore call this function whenever the graph structure is changed (e.g. by adding/removing edges or vertices).

#### Parameters

<i>modified</i>	If <code>true</code> the graph is modified, otherwise reset the status.
-----------------	---

Definition at line 359 of file graph.h.

References `_graph_modified`.

Referenced by `teb::TebController< p, q >::activateControlBounds()`, `teb::TebController< p, q >::activateObjectiveQuadraticForm()`, `teb::TebController< p, q >::activateObjectiveTimeOptimal()`, `teb::TebController< p, q >::activateStateBounds()`, `teb::TebController< p, q >::setFixedDt()`, `teb::TebController< p, q >::setFixedGoal()`, `teb::TebController< p, q >::setFixedStart()`, `teb::TebController< p, q >::setGoalStatesFixedOrUnfixed()`, and `teb::TebController< p, q >::setSystemDynamics()`.



#### 16.18.3.16 `EdgeContainer& teb::HyperGraph::objectives ( ) [inline]`

Objectives (subclasses of [BaseEdge](#)) are inserted within [TebController::buildOptimizationGraph\(\)](#) and [TebController::customOptimizationGraph\(\)](#) for instance.

##### Returns

Reference to the objective container.

Definition at line 229 of file `graph.h`.

References `_objectives`.

Referenced by `teb::BaseSolver::solve()`.

#### 16.18.3.17 `const EdgeContainer& teb::HyperGraph::objectives ( ) const [inline]`

Definition at line 231 of file `graph.h`.

References `_objectives`.

### 16.18.4 Member Data Documentation

#### 16.18.4.1 `VertexContainer teb::HyperGraph::_active_vertices [protected]`

Definition at line 216 of file `graph.h`.

Referenced by `activeVertices()`, `addActiveVertex()`, and `clearActiveVertices()`.

#### 16.18.4.2 `EdgeContainer teb::HyperGraph::_constraints_eq [protected]`

Definition at line 214 of file `graph.h`.

Referenced by `addEdgeEquality()`, `clearEdges()`, and `equalities()`.

#### 16.18.4.3 `EdgeContainer teb::HyperGraph::_constraints_ineq [protected]`

Definition at line 215 of file `graph.h`.

Referenced by `addEdgeInequality()`, `clearEdges()`, and `inequalities()`.

#### 16.18.4.4 `bool teb::HyperGraph::_graph_modified = true [protected]`

Definition at line 218 of file `graph.h`.

Referenced by `clearActiveVertices()`, `clearEdges()`, `isGraphModified()`, and `notifyGraphModified()`.

#### 16.18.4.5 `EdgeContainer teb::HyperGraph::_objectives [protected]`

Definition at line 213 of file `graph.h`.

Referenced by `addEdgeObjective()`, `clearEdges()`, and `objectives()`.

The documentation for this class was generated from the following file:

- [graph.h](#)

## 16.19 teb::JacobianWorkspace Class Reference

Memory management and accessor functions for the Block-Jacobians (for each edge).

```
#include <workspaces.h>
```

### Public Types

- using [WorkspaceMatrix](#) = std::vector< Eigen::MatrixXd, Eigen::aligned\_allocator< Eigen::MatrixXd >>  
*Typedef for the BlockJacobian container.*

### Public Member Functions

- [JacobianWorkspace](#) ()  
*Empty constructor.*
- [~JacobianWorkspace](#) ()  
*Empty destructor.*
- template<typename T >  
bool [allocate](#) (int edge\_dim, const T &vertices\_dim)  
*Access Block-Jacobian for TebVertex with index vertex\_idx.*
- Eigen::MatrixXd & [getWorkspace](#) (int vertex\_idx)  
*Access Block-Jacobian for TebVertex with index vertex\_idx.*

### Protected Attributes

- [WorkspaceMatrix \\_workspace](#)  
*Block-Jacobian container.*

#### 16.19.1 Detailed Description

This class controls the memory management of Block-Jacobians that are calculated for each edge. Each edge instantiates its own [JacobianWorkspace](#) and allocates the memory required to store all entries. Since the size depends on the type and number of attached vertices and the error dimension, it is not known a-priori. This workspace helps managing memory and accessing elements at run-time.

#### Remarks

For each vertex attached to a subclass of [BaseEdge](#), a single Jacobian of the size [ [BaseEdge::Dimension](#) x [TebVertex::dimension\(\)](#) ] is required.

Assume that a vertex has the following unfixed variables: [x1, x2, x3] and an edge has the cost values [v1, v2]:

$$\mathbf{J}_{block} = \begin{bmatrix} \partial_{x1} v1 & \partial_{x2} v1 & \partial_{x3} v1 \\ \partial_{x1} v2 & \partial_{x2} v2 & \partial_{x3} v2 \end{bmatrix}$$

Let the vertex be the first one of the corresponding edge. The above Block-Jacobian can be stored from an edge (after calling [allocate\(\)](#) once) as follows:

```
this->getWorkspace(0) = J_block; // exchange "this" by a pointer/ref to a valid workspace
instance with allocated and reserved memory.
```

**Todo** [TebVertex::dimension\(\)](#) is used at the moment. Acutally only a number of [TebVertex::dimensionFree\(\)](#) non-zeros may appear. Test if it speeds up the optimization to skip those values by using the fixed-mask. But for common MPC problems only the first or/and the final state are fixed partially.

## Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

## See Also

[BaseEdge](#), [HessianWorkspace](#)

Definition at line 48 of file workspaces.h.

## 16.19.2 Member Typedef Documentation

16.19.2.1 `using teb::JacobianWorkspace::WorkspaceMatrix = std::vector<Eigen::MatrixXd, Eigen::aligned_allocator<Eigen::MatrixXd>>`

A single block-matrix per attached TebVertex is required.

Definition at line 52 of file workspaces.h.

## 16.19.3 Constructor &amp; Destructor Documentation

16.19.3.1 `teb::JacobianWorkspace::JacobianWorkspace ( )` `[inline]`

Definition at line 54 of file workspaces.h.

16.19.3.2 `teb::JacobianWorkspace::~~JacobianWorkspace ( )` `[inline]`

Definition at line 55 of file workspaces.h.

## 16.19.4 Member Function Documentation

16.19.4.1 `template<typename T> bool teb::JacobianWorkspace::allocate ( int edge_dim, const T & vertices_dim )` `[inline]`

Allocate memory for all block-Jacobians.

## Parameters

<i>edge_dim</i>	Dimension of the edge. Typically <a href="#">BaseEdge::Dimension()</a> .
<i>vertices_dim</i>	Vector of dimensions of the attached vertices. Typically <a href="#">TebVertex::dimension()</a> each.

## Template Parameters

<i>T</i>	storage class (std::vector, std::array, std::deque)
----------	---

## Return values

<i>true</i>	Status flag. Currently it returns only <code>true</code> .
-------------	--

Definition at line 65 of file workspaces.h.

References [\\_workspace](#).

Referenced by `teb::BaseEdge< 1, FUNCT_TYPE::LINEAR, TimeDiff >::allocateMemory()`, and `teb::BaseEdge< D, FuncType >::allocateMemory()`.

16.19.4.2 `Eigen::MatrixXd& teb::JacobianWorkspace::getWorkspace ( int vertex_idx )` `[inline]`

**Parameters**

<i>vertex_idx</i>	Index of the corresponding vertex.
-------------------	------------------------------------

**Returns**

Reference to Block-Jacobian of vertex with index `vertex_idx`.

Definition at line 82 of file `workspaces.h`.

References `_workspace`.

Referenced by `teb::BaseEdge< D, FuncType, Vertices >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::EdgeMinimizeTime::computeJacobian()`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::computeJacobian()`, `teb::EdgeQuadraticForm< p, q >::computeJacobian()`, and `teb::BaseEdge< D, FuncType >::computeJacobian()`.

**16.19.5 Member Data Documentation****16.19.5.1 WorkspaceMatrix** `teb::JacobianWorkspace::_workspace` [protected]

Definition at line 82 of file `workspaces.h`.

Referenced by `allocate()`, and `getWorkspace()`.

The documentation for this class was generated from the following file:

- [workspaces.h](#)

**16.20 teb::Config::Optim::Solver::NonlinearProgram::LineSearch Struct Reference**

Settings for the line-search algorithm (force merit function descent)

```
#include <config.h>
```

**Public Attributes**

- double [sigma](#) = 0.5
- double [beta](#) = 0.5
- double [alpha\\_init](#) = 0.1

**16.20.1 Detailed Description**

Definition at line 105 of file `config.h`.

**16.20.2 Member Data Documentation****16.20.2.1** double `teb::Config::Optim::Solver::NonlinearProgram::LineSearch::alpha_init` = 0.1

Definition at line 109 of file `config.h`.

Referenced by `teb::SolverSQPDense::initSolverWorkspace()`.

16.20.2.2 double teb::Config::Optim::Solver::NonlinearProgram::LineSearch::beta = 0.5

Definition at line 108 of file config.h.

Referenced by teb::SolverSQPDense::solveImpl().

16.20.2.3 double teb::Config::Optim::Solver::NonlinearProgram::LineSearch::sigma = 0.5

Definition at line 107 of file config.h.

Referenced by teb::SolverSQPDense::solveImpl().

The documentation for this struct was generated from the following file:

- [config.h](#)

## 16.21 teb::Config::Optim::Solver::Lsq Struct Reference

Configurations especially for least-squares-solvers.

```
#include <config.h>
```

### Public Attributes

- double [soft\\_constr\\_epsilon](#) = 0.  
*Add a bigger margin to soft-constraints.*
- double [weight\\_equalities](#) = 2  
*Soft-constraint weight for equality constraints.*
- double [weight\\_inequalities](#) = 2  
*Soft-constraint weight for inequality constraints.*
- double [weight\\_adaptation\\_factor](#) = 2  
*Factor for soft-constraint weight adaption (see [BaseSolverLeastSquares::adaptWeights\(\)](#)).*

### 16.21.1 Detailed Description

Definition at line 83 of file config.h.

### 16.21.2 Member Data Documentation

16.21.2.1 double teb::Config::Optim::Solver::Lsq::soft\_constr\_epsilon = 0.

**Bug** Do not use at the moment, because it is not implemented correctly

Examples:

[rocket\\_system.cpp](#).

Definition at line 85 of file config.h.

Referenced by teb::SolverLevenbergMarquardtEigenDense::buildJacobian(), teb::SolverLevenbergMarquardtEigenSparse::buildJacobian(), and teb::BaseSolverLeastSquares::buildValueVector().

16.21.2.2 `double teb::Config::Optim::Solver::Lsq::weight_adaptation_factor = 2`

Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Definition at line 88 of file `config.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`.

16.21.2.3 `double teb::Config::Optim::Solver::Lsq::weight_equalities = 2`

Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), and [mobile\\_robot\\_teb.cpp](#).

Definition at line 86 of file `config.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`.

16.21.2.4 `double teb::Config::Optim::Solver::Lsq::weight_inequalities = 2`

Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Definition at line 87 of file `config.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`.

The documentation for this struct was generated from the following file:

- [config.h](#)

## 16.22 `teb::Config::Optim::Solver::Nlopt` Struct Reference

Configurations especially for the nlopt solver wrapper.

```
#include <config.h>
```

### Public Attributes

- `NloptAlgorithms algorithm = NloptAlgorithms::SLSQP`
- `double tolerance_equalities = 0.001`  
*Stop optimization depending of the equality constraint satisfaction tolerance.*
- `double tolerance_inequalities = 0.001`  
*Stop optimization depending of the inequality constraint satisfaction tolerance.*
- `double stopping_criteria_ftol_abs = 0.0001`  
*Stop optimization depending on the absolute value of the objective function.*
- `double stopping_criteria_ftol_rel = 0.0001`  
*Stop optimization depending on the relative change of the objective function value.*
- `double stopping_criteria_xtol_abs = 0.0001`  
*Stop optimization depending on the absolute values of the optimization vector.*
- `double stopping_criteria_xtol_rel = 0.0001`  
*Stop optimization depending on the relative change of values of the optimization vector.*
- `double max_optimization_time = 5`  
*Stop optimization after a given duration in seconds.*

### 16.22.1 Detailed Description

Definition at line 115 of file config.h.

### 16.22.2 Member Data Documentation

#### 16.22.2.1 NloptAlgorithms teb::Config::Optim::Solver::Nlopt::algorithm = NloptAlgorithms::SLSQP

Definition at line 117 of file config.h.

#### 16.22.2.2 double teb::Config::Optim::Solver::Nlopt::max\_optimization\_time = 5

Definition at line 126 of file config.h.

#### 16.22.2.3 double teb::Config::Optim::Solver::Nlopt::stopping\_criteria\_ftol\_abs = 0.0001

Definition at line 122 of file config.h.

#### 16.22.2.4 double teb::Config::Optim::Solver::Nlopt::stopping\_criteria\_ftol\_rel = 0.0001

Definition at line 123 of file config.h.

#### 16.22.2.5 double teb::Config::Optim::Solver::Nlopt::stopping\_criteria\_xtol\_abs = 0.0001

Definition at line 124 of file config.h.

#### 16.22.2.6 double teb::Config::Optim::Solver::Nlopt::stopping\_criteria\_xtol\_rel = 0.0001

Definition at line 125 of file config.h.

#### 16.22.2.7 double teb::Config::Optim::Solver::Nlopt::tolerance\_equalities = 0.001

Definition at line 119 of file config.h.

#### 16.22.2.8 double teb::Config::Optim::Solver::Nlopt::tolerance\_inequalities = 0.001

Definition at line 120 of file config.h.

The documentation for this struct was generated from the following file:

- [config.h](#)

## 16.23 teb::Config::Optim::Solver::NonlinearProgram Struct Reference

Settings for constrained optimization solver (nonlinear program solver)

```
#include <config.h>
```

## Classes

- struct [Hessian](#)

*Configurations for the hessian of the lagrangian calculation.*

- struct [LineSearch](#)

*Settings for the line-search algorithm (force merit function descent)*

## Public Attributes

- struct

[teb::Config::Optim::Solver::NonlinearProgram::Hessian](#) `hessian`

*Configurations for the hessian of the lagrangian calculation.*

- struct

[teb::Config::Optim::Solver::NonlinearProgram::LineSearch](#) `linesearch`

*Settings for the line-search algorithm (force merit function descent)*

### 16.23.1 Detailed Description

Definition at line 93 of file `config.h`.

### 16.23.2 Member Data Documentation

**16.23.2.1** struct `teb::Config::Optim::Solver::NonlinearProgram::Hessian` `teb::Config::Optim::Solver::NonlinearProgram::hessian`

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

**16.23.2.2** struct `teb::Config::Optim::Solver::NonlinearProgram::LineSearch` `teb::Config::Optim::Solver::NonlinearProgram::linesearch`

Referenced by `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

The documentation for this struct was generated from the following file:

- [config.h](#)

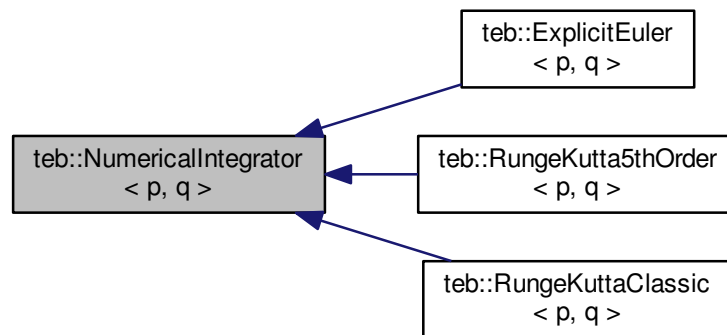
## 16.24 `teb::NumericalIntegrator< p, q >` Class Template Reference

Interface for numerical integration methods used in simulation.

```
#include <integrators.h>
```



Inheritance diagram for teb::NumericalIntegrator< p, q >:



## Public Types

- using [StateVector](#) = Eigen::Matrix< double, p, 1 >  
*Typedef for state vector with p states [p x 1].*
- using [ControlVector](#) = Eigen::Matrix< double, q, 1 >  
*Typedef for control input vector with q controls [q x 1].*

## Public Member Functions

- [NumericalIntegrator](#) ()  
*Empty Constructor.*
- virtual [~NumericalIntegrator](#) ()  
*Empty Destructor.*
- virtual [StateVector integrate](#) (const Eigen::Ref< const [StateVector](#) > &x\_k, const Eigen::Ref< const [ControlVector](#) > &u\_k, [SystemDynamics](#)< p, q > \*system\_equation, double dt) const =0  
*Integrate a nonlinear state-space model*

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

### 16.24.1 Detailed Description

```
template<int p, int q>class teb::NumericalIntegrator< p, q >
```

This class defines the interface for numerical integration schemes.

**Todo** Implement and test Runge Kutta with derivatives <http://www.emis.de/journals/EJD-E/conf-proc/02/g1/goeken.pdf>

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

## Template Parameters

$p$	Number of state variables
$q$	Number of input variables

Definition at line 36 of file integrators.h.

## 16.24.2 Member Typedef Documentation

16.24.2.1 `template<int p, int q> using teb::NumericalIntegrator< p, q >::ControlVector = Eigen::Matrix<double,q,1>`

Definition at line 43 of file integrators.h.

16.24.2.2 `template<int p, int q> using teb::NumericalIntegrator< p, q >::StateVector = Eigen::Matrix<double,p,1>`

Definition at line 41 of file integrators.h.

## 16.24.3 Constructor &amp; Destructor Documentation

16.24.3.1 `template<int p, int q> teb::NumericalIntegrator< p, q >::NumericalIntegrator ( ) [inline]`

Definition at line 45 of file integrators.h.

16.24.3.2 `template<int p, int q> virtual teb::NumericalIntegrator< p, q >::~~NumericalIntegrator ( ) [inline], [virtual]`

Definition at line 46 of file integrators.h.

## 16.24.4 Member Function Documentation

16.24.4.1 `template<int p, int q> virtual StateVector teb::NumericalIntegrator< p, q >::integrate ( const Eigen::Ref< const StateVector > & x_k, const Eigen::Ref< const ControlVector > & u_k, SystemDynamics< p, q > * system_equation, double dt ) const [pure virtual]`

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t.  $\mathbf{x}$ . In particular, it relies on discrete states. The method calculates  $\mathbf{x}_{k+1}$  based on  $\mathbf{x}_k$ ,  $\mathbf{u}_k$  and the duration  $\Delta T$ . The dedicated system dynamics equations may be specified using a [SystemDynamics](#) object.

## Parameters

$x\_k$	Current state vector $\mathbf{x}_k$ [p x 1]
$u\_k$	Current state vector $\mathbf{u}_k$ [q x 1]
<i>system_equation</i>	Pointer to the <a href="#">SystemDynamics</a> object that stores the system equations.
<i>dt</i>	Time interval for the integration

## Returns

Integrated state vector  $\mathbf{x}_{k+1}$  [p x 1]

Implemented in [teb::RungeKutta5thOrder< p, q >](#), [teb::RungeKuttaClassic< p, q >](#), and [teb::ExplicitEuler< p, q >](#).

The documentation for this class was generated from the following file:

- [integrators.h](#)

## 16.25 teb::Config::Optim Struct Reference

Configurations related to the trajectory optimization.

```
#include <config.h>
```

### Classes

- struct [Solver](#)  
*Configurations related to solvers.*

### Public Attributes

- struct [teb::Config::Optim::Solver](#) solver  
*Configurations related to solvers.*
- bool [force\\_rebuild\\_optim\\_graph](#) = false  
*Disable hot-starting from previous graph structures even if no changes to the graph are made.*

#### 16.25.1 Detailed Description

Definition at line 75 of file config.h.

#### 16.25.2 Member Data Documentation

##### 16.25.2.1 bool teb::Config::Optim::force\_rebuild\_optim\_graph = false

Definition at line 133 of file config.h.

##### 16.25.2.2 struct teb::Config::Optim::Solver teb::Config::Optim::solver

Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Referenced by [teb::BaseSolverLeastSquares::adaptWeights\(\)](#), [teb::SolverLevenbergMarquardtEigenDense::buildJacobian\(\)](#), [teb::SolverLevenbergMarquardtEigenSparse::buildJacobian\(\)](#), [teb::BaseSolverLeastSquares::buildValueVector\(\)](#), [teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian\(\)](#), [teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS\(\)](#), [teb::BaseSolverNonlinearProgramDense::initHessianBFGS\(\)](#), [teb::SolverSQPDense::initSolverWorkspace\(\)](#), and [teb::SolverSQPDense::solveImpl\(\)](#).

The documentation for this struct was generated from the following file:

- [config.h](#)

## 16.26 teb::PlotOptions Struct Reference

Customize plots and figures.

```
#include <teb_plotter.h>
```

## Public Attributes

- `std::string title = ""`  
*Window title.*
- `bool legend = false`  
*Show legend if `true`.*
- `std::vector< std::string > legend_entries`  
*Vector of strings defining the names for the legend.*
- `bool skip_last_value_right_column = false`
- `std::vector< std::string > ylabels`  
*< If `true`, the last value of the right column for each plot will be ignored (use to suppress plotting the invalid control for time `n`)*

### 16.26.1 Detailed Description

This option class can be used to customize figures and plots, if they are supported by the plot function.

**Todo** Extend functionality

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

Definition at line 27 of file `teb_plotter.h`.

### 16.26.2 Member Data Documentation

#### 16.26.2.1 `bool teb::PlotOptions::legend = false`

Definition at line 30 of file `teb_plotter.h`.

Referenced by `teb::TebPlotter::plotTwoCol()`, `teb::Simulator< p, q >::simClosedLoop()`, `teb::Simulator< p, q >::simOpenAndClosedLoop()`, and `teb::Simulator< p, q >::simOpenLoop()`.

#### 16.26.2.2 `std::vector<std::string> teb::PlotOptions::legend_entries`

##### Remarks

legend should be enabled.

Definition at line 31 of file `teb_plotter.h`.

Referenced by `teb::TebPlotter::plotTwoCol()`, `teb::Simulator< p, q >::simClosedLoop()`, `teb::Simulator< p, q >::simOpenAndClosedLoop()`, and `teb::Simulator< p, q >::simOpenLoop()`.

#### 16.26.2.3 `bool teb::PlotOptions::skip_last_value_right_column = false`

Definition at line 32 of file `teb_plotter.h`.

Referenced by `teb::TebPlotter::plotTwoCol()`, `teb::Simulator< p, q >::simClosedLoop()`, `teb::Simulator< p, q >::simOpenAndClosedLoop()`, and `teb::Simulator< p, q >::simOpenLoop()`.

16.26.2.4 `std::string teb::PlotOptions::title = ""`

Definition at line 29 of file `teb_plotter.h`.

Referenced by `teb::TebPlotter::plotTwoCol()`, `teb::Simulator< p, q >::simClosedLoop()`, `teb::Simulator< p, q >::simOpenAndClosedLoop()`, and `teb::Simulator< p, q >::simOpenLoop()`.

16.26.2.5 `std::vector<std::string> teb::PlotOptions::ylabels`

Vector of strings to define the y labels (Column-Major Order) - leave blank in order to use default names

Definition at line 33 of file `teb_plotter.h`.

Referenced by `teb::TebPlotter::plotTwoCol()`.

The documentation for this struct was generated from the following file:

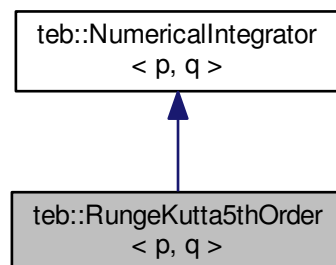
- [teb\\_plotter.h](#)

16.27 `teb::RungeKutta5thOrder< p, q >` Class Template Reference

Runge Kutta method (fifth order, slightly modified)

```
#include <integrators.h>
```

Inheritance diagram for `teb::RungeKutta5thOrder< p, q >`:



## Public Types

- using `StateVector` = typename `NumericalIntegrator< p, q >::StateVector`  
*Typedef for state vector with p states [p x 1].*
- using `ControlVector` = typename `NumericalIntegrator< p, q >::ControlVector`  
*Typedef for control input vector with q controls [q x 1].*

## Public Member Functions

- `RungeKutta5thOrder()`  
*Empty Constructor.*
- virtual `~RungeKutta5thOrder()`

*Empty Destructor.*

- virtual [StateVector](#) `integrate` (const Eigen::Ref< const [StateVector](#) > &x\_k, const Eigen::Ref< const [ControlVector](#) > &u\_k, [SystemDynamics](#)< p, q > \*system\_equation, double dt) const

*Integrate a nonlinear state-space model*

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

### 16.27.1 Detailed Description

```
template<int p, int q>class teb::RungeKutta5thOrder< p, q >
```

For more information please refer to <http://www.jstor.org/stable/pdfplus/2027775.-pdf?acceptTC=true&jpdConfirm=true> (Equation 2)

Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

Template Parameters

<i>p</i>	Number of state variables
<i>q</i>	Number of input variables

Definition at line 157 of file integrators.h.

### 16.27.2 Member Typedef Documentation

16.27.2.1 `template<int p, int q> using teb::RungeKutta5thOrder< p, q >::ControlVector = typename NumericalIntegrator<p,q>::ControlVector`

Definition at line 163 of file integrators.h.

16.27.2.2 `template<int p, int q> using teb::RungeKutta5thOrder< p, q >::StateVector = typename NumericalIntegrator<p,q>::StateVector`

Definition at line 161 of file integrators.h.

### 16.27.3 Constructor & Destructor Documentation

16.27.3.1 `template<int p, int q> teb::RungeKutta5thOrder< p, q >::RungeKutta5thOrder ( ) [inline]`

Definition at line 166 of file integrators.h.

16.27.3.2 `template<int p, int q> virtual teb::RungeKutta5thOrder< p, q >::~~RungeKutta5thOrder ( ) [inline], [virtual]`

Definition at line 167 of file integrators.h.

### 16.27.4 Member Function Documentation

16.27.4.1 `template<int p, int q> virtual StateVector teb::RungeKutta5thOrder< p, q >::integrate ( const Eigen::Ref< const StateVector > & x_k, const Eigen::Ref< const ControlVector > & u_k, SystemDynamics< p, q > * system_equation, double dt ) const` `[inline], [virtual]`

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t.  $\mathbf{x}$ . In particular, it relies on discrete states. The method calculates  $\mathbf{x}_{k+1}$  based on  $\mathbf{x}_k$ ,  $\mathbf{u}_k$  and the duration  $\Delta T$ . The dedicated system dynamics equations may be specified using a [SystemDynamics](#) object.

#### Parameters

<code>x_k</code>	Current state vector $\mathbf{x}_k$ [p x 1]
<code>u_k</code>	Current state vector $\mathbf{u}_k$ [q x 1]
<code>system_equation</code>	Pointer to the <a href="#">SystemDynamics</a> object that stores the system equations.
<code>dt</code>	Time interval for the integration

#### Returns

Integrated state vector  $\mathbf{x}_{k+1}$  [p x 1]

Implements [teb::NumericalIntegrator< p, q >](#).

Definition at line 171 of file `integrators.h`.

References [teb::SystemDynamics< p, q >::stateSpaceModel\(\)](#).

The documentation for this class was generated from the following file:

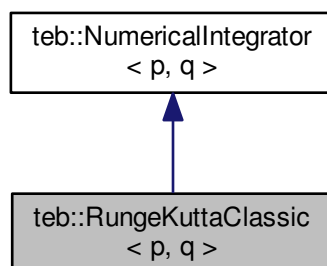
- [integrators.h](#)

## 16.28 teb::RungeKuttaClassic< p, q > Class Template Reference

Classic Runge Kutta method (fourth order)

```
#include <integrators.h>
```

Inheritance diagram for `teb::RungeKuttaClassic< p, q >`:



#### Public Types

- using [StateVector](#) = typename [NumericalIntegrator< p, q >::StateVector](#)

*Typedef for state vector with  $p$  states [ $p \times 1$ ].*

- using `ControlVector` = typename `NumericalIntegrator`<  $p$ ,  $q$  >::`ControlVector`

*Typedef for control input vector with  $q$  controls [ $q \times 1$ ].*

## Public Member Functions

- `RungeKuttaClassic` ()  
*Empty Constructor.*
- virtual `~RungeKuttaClassic` ()  
*Empty Destructor.*
- virtual `StateVector integrate` (const Eigen::Ref< const `StateVector` > & $x_k$ , const Eigen::Ref< const `ControlVector` > & $u_k$ , `SystemDynamics`<  $p$ ,  $q$  > \* $system\_equation$ , double  $dt$ ) const  
*Integrate a nonlinear state-space model*

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

### 16.28.1 Detailed Description

```
template<int p, int q>class teb::RungeKuttaClassic< p, q >
```

For more information please refer to [http://en.wikipedia.org/wiki/Runge-Kutta\\_methods](http://en.wikipedia.org/wiki/Runge-Kutta_methods)

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### Template Parameters

$p$	Number of state variables
$q$	Number of input variables

Definition at line 115 of file `integrators.h`.

### 16.28.2 Member Typedef Documentation

16.28.2.1 `template<int p, int q> using teb::RungeKuttaClassic< p, q >::ControlVector = typename NumericalIntegrator<p,q>::ControlVector`

Definition at line 121 of file `integrators.h`.

16.28.2.2 `template<int p, int q> using teb::RungeKuttaClassic< p, q >::StateVector = typename NumericalIntegrator<p,q>::StateVector`

Definition at line 119 of file `integrators.h`.

### 16.28.3 Constructor & Destructor Documentation

16.28.3.1 `template<int p, int q> teb::RungeKuttaClassic< p, q >::RungeKuttaClassic ( ) [inline]`

Definition at line 124 of file `integrators.h`.



16.28.3.2 `template<int p, int q> virtual teb::RungeKuttaClassic< p, q >::~RungeKuttaClassic ( ) [inline], [virtual]`

Definition at line 125 of file integrators.h.

## 16.28.4 Member Function Documentation

16.28.4.1 `template<int p, int q> virtual StateVector teb::RungeKuttaClassic< p, q >::integrate ( const Eigen::Ref< const StateVector > & x_k, const Eigen::Ref< const ControlVector > & u_k, SystemDynamics< p, q > * system_equation, double dt ) const [inline], [virtual]`

Integrate a nonlinear state-space model

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

w.r.t.  $\mathbf{x}$ . In particular, it relies on discrete states. The method calculates  $\mathbf{x}_{k+1}$  based on  $\mathbf{x}_k$ ,  $\mathbf{u}_k$  and the duration  $\Delta T$ . The dedicated system dynamics equations may be specified using a [SystemDynamics](#) object.

Parameters

$x\_k$	Current state vector $\mathbf{x}_k$ [p x 1]
$u\_k$	Current state vector $\mathbf{u}_k$ [q x 1]
<i>system_equation</i>	Pointer to the <a href="#">SystemDynamics</a> object that stores the system equations.
<i>dt</i>	Time interval for the integration

Returns

Integrated state vector  $\mathbf{x}_{k+1}$  [p x 1]

Implements [teb::NumericalIntegrator< p, q >](#).

Definition at line 129 of file integrators.h.

References [teb::SystemDynamics< p, q >::stateSpaceModel\(\)](#).

The documentation for this class was generated from the following file:

- [integrators.h](#)

## 16.29 teb::SimResults Class Reference

Simulation results are stored and combined in this container class.

```
#include <simulator.h>
```

### Classes

- struct [TimeSeries](#)

*Store measurements of dynamic systems (states and control inputs) w.r.t.*

### Public Attributes

- `std::vector< TimeSeries > series`

*Container for multiple time series (e.g. to perform store different experiments or closed- and open-loop sim results).*

## Protected Member Functions

- void [conservativeResize](#) (unsigned int id, int p, int q, int n)  
*Resize time series, but preserve data with indices below  $n$ .*
- void [allocateMemory](#) (unsigned int id, int p, int q, int n, bool set\_zero=true)  
*Allocate memory for all matrices and vectors of the underlying [TimeSeries](#) object.*

## Static Protected Member Functions

- static void [conservativeResize](#) ([TimeSeries](#) &data, int p, int q, int n)  
*Resize time series, but preserve data with indices below  $n$ .*
- static void [allocateMemory](#) ([TimeSeries](#) &data, int p, int q, int n, bool set\_zero=true)  
*Allocate memory for all matrices and vectors of the underlying [TimeSeries](#) object.*

## Friends

- template<int pf, int qf>  
class [Simulator](#)

## 16.29.1 Detailed Description

### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

### Template Parameters

$p$	Number of state variables
$q$	Number of input variables

Definition at line 26 of file simulator.h.

## 16.29.2 Member Function Documentation

**16.29.2.1** static void `teb::SimResults::allocateMemory ( TimeSeries & data, int  $p$ , int  $q$ , int  $n$ , bool set_zero = true )`  
[inline], [static], [protected]

This method can be used to resize the underlying matrices without preserving data.

### See Also

[conservativeResize\(\)](#)

### Parameters

<i>data</i>	<a href="#">TimeSeries</a> object for which the memory should be allocated.
$p$	Size of the state vector.
$q$	Size of the control input vector.
$n$	Number of discrete samples.
<i>set_zero</i>	if <code>true</code> , all matrices and vectors are initialized to zero.

Definition at line 89 of file simulator.h.

References `teb::SimResults::TimeSeries::controls`, `teb::SimResults::TimeSeries::states`, and `teb::SimResults::TimeSeries::time`.

Referenced by `allocateMemory()`, and `teb::Simulator< p, q >::simOpenLoop()`.

**16.29.2.2** `void teb::SimResults::allocateMemory ( unsigned int id, int p, int q, int n, bool set_zero = true ) [inline], [protected]`

This method can be used to resize the underlying matrices without preserving data.

See Also

[conservativeResize\(\)](#)

Parameters

<i>id</i>	Id/index of the time series object stored in <a href="#">SimResults::series</a>
<i>p</i>	Size of the state vector.
<i>q</i>	Size of the control input vector.
<i>n</i>	Number of discrete samples.
<i>set_zero</i>	if <code>true</code> , all matrices and vectors are initialized to zero.

Definition at line 118 of file simulator.h.

References `allocateMemory()`, and `series`.

**16.29.2.3** `static void teb::SimResults::conservativeResize ( TimeSeries & data, int p, int q, int n ) [inline], [static], [protected]`

Parameters

<i>data</i>	<a href="#">TimeSeries</a> object which should be resized.
<i>p</i>	Size of the state vector.
<i>q</i>	Size of the control input vector.
<i>n</i>	New number of discrete samples.

Definition at line 56 of file simulator.h.

References `teb::SimResults::TimeSeries::controls`, `teb::SimResults::TimeSeries::states`, and `teb::SimResults::TimeSeries::time`.

Referenced by `conservativeResize()`, and `teb::Simulator< p, q >::simClosedLoop()`.

**16.29.2.4** `void teb::SimResults::conservativeResize ( unsigned int id, int p, int q, int n ) [inline], [protected]`

Parameters

<i>id</i>	Id/index of the time series object stored in <a href="#">SimResults::series</a>
<i>p</i>	Size of the state vector.
<i>q</i>	Size of the control input vector.
<i>n</i>	New number of discrete samples.

Definition at line 70 of file simulator.h.

References `conservativeResize()`, and `series`.

## 16.29.3 Friends And Related Function Documentation

**16.29.3.1** `template<int pf, int qf> friend class Simulator [friend]`

Definition at line 29 of file simulator.h.

## 16.29.4 Member Data Documentation

#### 16.29.4.1 `std::vector<TimeSeries> teb::SimResults::series`

Definition at line 45 of file `simulator.h`.

Referenced by `allocateMemory()`, `conservativeResize()`, `teb::Simulator< p, q >::simClosedLoop()`, `teb::Simulator< p, q >::simOpenAndClosedLoop()`, and `teb::Simulator< p, q >::simOpenLoop()`.

The documentation for this class was generated from the following file:

- [simulator.h](#)

## 16.30 `teb::Simulator< p, q >` Class Template Reference

[Simulator](#) for dynamic systems controlled by a [TebController](#).

```
#include <simulator.h>
```

### Public Types

- using [StateVector](#) = `Eigen::Matrix< double, p, 1 >`  
*Typedef for state vector with p states [p x 1].*
- using [ControlVector](#) = `Eigen::Matrix< double, q, 1 >`  
*Typedef for control input vector with q controls [q x 1].*
- using [IntegratorPtr](#) = `std::unique_ptr< NumericalIntegrator< p, q >>`  
*Typedef for a smart pointer that points to the preferred numerical integration method object.*

### Public Member Functions

- [Simulator](#) ([BaseController](#) \*controller, [SystemDynamics](#)< p, q > &system, [TebPlotter](#) \*plotter=nullptr)  
*Construct the simulator by accepting a [TebController](#) object, a [SystemDynamics](#) object and the [TebPlotter](#) object.*
- [Simulator](#) ([SystemDynamics](#)< p, q > &system, [TebPlotter](#) \*plotter=nullptr)  
*Construct the simulator using a [SystemDynamics](#) object and the [TebPlotter](#) object.*
- [~Simulator](#) ()  
*Empty Destructor.*
- void [setSampleTime](#) (double sample\_time)  
*Set the sample time for the simulation.*
- double [sampleTime](#) ()  
*Get the sample time of the simulator.*
- void [setPlotter](#) ([TebPlotter](#) \*plotter)  
*Pass a [TebPlotter](#) object to the [Simulator](#) class in order to enable visualization.*
- void [setIntegrator](#) ([IntegratorPtr](#) integrator)  
*Set a custom numerical integration method for simulation as subclass of [NumericalIntegrator](#).*
- void [setIntegrator](#) ([NumericalIntegrators](#) int\_type)  
*Set a predined numerical integration method for simulation (select from [teb::NumericalIntegrators](#))*
- `std::unique_ptr< SimResults > simOpenLoop (const double *const x0, const double *const xf, bool plot=true, BaseController *ctrl=nullptr)`  
*Perform an open-loop simulation of the system in combination with the [TebController](#).*
- `std::unique_ptr< SimResults > simClosedLoop (const double *const x0, const double *const xf, double sim_time, bool manual_stepping=false, bool plot_result=true, bool plot_steps=true, BaseController *ctrl=nullptr)`  
*Perform a closed-loop simulation of the system in combination with the [TebController](#).*
- `std::unique_ptr< SimResults > simOpenAndClosedLoop (const double *const x0, const double *const xf, double sim_time, bool manual_stepping=false, bool plot_steps=true)`

Perform both an open-loop and a closed-loop simulation of the system in combination with the [TebController](#).

- void [simOpenLoop](#) (std::vector< std::pair< [BaseController](#) \*, std::string >> &controllers, const double \*const x0, const double \*const xf, bool plot=true)

Perform an open-loop simulation of the system for different [TebControllers](#).

- void [simClosedLoop](#) (std::vector< std::pair< [BaseController](#) \*, std::string >> &controllers, const double \*const x0, const double \*const xf, double sim\_time, bool plot\_result=true, bool plot\_steps=true)

Perform a closed-loop simulation of the system for different [TebControllers](#).

- [StateVector](#) [systemStep](#) (const Eigen::Ref< const [StateVector](#) > &x\_k, const Eigen::Ref< const [ControlVector](#) > &u\_k, double dt)
- void [setControlInputSaturation](#) (const double \*u\_min, const double \*u\_max)

Set bounds on the control inputs for simulation (saturation).

- void [setPreStepCallback](#) (const std::function< void([BaseController](#) \*, [SystemDynamics](#)< p, q > \*, [Simulator](#) \*)> &callback)

Set a pointer to a function that is called at the beginning of each control step.

- void [setPostStepCallback](#) (const std::function< void([BaseController](#) \*, [SystemDynamics](#)< p, q > \*, [Simulator](#) \*)> &callback)

Set a pointer to a function that is called after each control step.

- void [setPreSimCallback](#) (const std::function< void([BaseController](#) \*, [SystemDynamics](#)< p, q > \*, [Simulator](#) \*)> &callback)

Set a pointer to a function that is called at the beginning of each complete simulation.

## Protected Member Functions

- void [plotResults](#) (const std::vector< [SimResults::TimeSeries](#) > &time\_series, [teb::PlotOptions](#) \*options=nullptr) const

Plot a container of [SimResults::TimeSeries](#) objects.

- void [saturateControl](#) (Eigen::Ref< [ControlVector](#) > u)

Saturate the control inputs **u** specified by [setControlInputSaturation\(\)](#).

## Protected Attributes

- [BaseController](#) \* [\\_controller](#) = nullptr

Pointer to the Controller used for controlling the system.

- [SystemDynamics](#)< p, q > \* [\\_system](#) = nullptr

Pointer to the [SystemDynamics](#) object that specifies the state space equations of the dynamic system.

- [IntegratorPtr](#) [\\_integrator](#) = [IntegratorPtr](#)(new [RungeKutta5thOrder](#)<p,q>())

Numerical integration method (default: [RungeKutta5thOrder](#))

- [TebPlotter](#) \* [\\_plotter](#) = nullptr

Pointer to a [TebPlotter](#) object used for visualization.

- double [\\_sample\\_time](#) = -1.

Sample time for simulation (-1: inherit variable dt from teb)

- std::pair< [ControlVector](#), [ControlVector](#) > [\\_control\\_bounds](#) = std::make\_pair([ControlVector::Constant](#)(-INF), [ControlVector::Constant](#)(INF))

Store hard control input bounds  $u_{min}$  and  $u_{max}$  for simulation (saturation).

- std::function< void([BaseController](#) \*, [SystemDynamics](#)< p, q > \*, [Simulator](#) \*)> [\\_callback\\_step\\_pre](#)

Store a pointer to a function that is called at the beginning of each control step Use this function to make user-defined changes a-priori to each step.

- std::function< void([BaseController](#) \*, [SystemDynamics](#)< p, q > \*, [Simulator](#) \*)> [\\_callback\\_step\\_post](#)

Store a pointer to a function that is called after each control step Use this function to make user-defined changes a-posteriori to each step.

- `std::function< void(BaseController`  
`*, SystemDynamics< p, q >`  
`*, Simulator *)> _callback_sim_pre`

Store a pointer to a function that is called at the beginning of each simulation Use this function to make user-defined changes a-priori to the complete simulation.

## Friends

- `template<int pf, int qf>`  
`class SystemDynamics`
- `class SimResults`

### 16.30.1 Detailed Description

```
template<int p, int q>class teb::Simulator< p, q >
```

This simulator supports open-loop and closed-loop simulations for systems specified using the [SystemDynamics](#) object.

Results can be visualized using the [TebPlotter](#) class.

**Todo** Add disturbance models.

Add reference trajectory or multiple reference steps for closed-loop control.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### Template Parameters

<i>p</i>	Number of state variables
<i>q</i>	Number of input variables

#### Examples:

[integrator\\_system2.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Definition at line 145 of file simulator.h.

### 16.30.2 Member Typedef Documentation

16.30.2.1 `template<int p, int q> using teb::Simulator< p, q >::ControlVector = Eigen::Matrix<double, q, 1>`

Definition at line 157 of file simulator.h.

16.30.2.2 `template<int p, int q> using teb::Simulator< p, q >::IntegratorPtr = std::unique_ptr<Numerical-Integrator<p,q>>`

Definition at line 159 of file simulator.h.

16.30.2.3 `template<int p, int q> using teb::Simulator< p, q >::StateVector = Eigen::Matrix<double,p,1>`

Definition at line 155 of file `simulator.h`.

### 16.30.3 Constructor & Destructor Documentation

16.30.3.1 `template<int p, int q> teb::Simulator< p, q >::Simulator ( BaseController * controller, SystemDynamics< p, q > & system, TebPlotter * plotter = nullptr ) [inline]`

Parameters

<i>controller</i>	Reference to the Controller
<i>system</i>	Reference to the underlying dynamic system.
<i>plotter</i>	Pointer to a <a href="#">TebPlotter</a> object that enables visualization.

Definition at line 169 of file `simulator.h`.

16.30.3.2 `template<int p, int q> teb::Simulator< p, q >::Simulator ( SystemDynamics< p, q > & system, TebPlotter * plotter = nullptr ) [inline]`

Usually you want to use the other constructor and pass a controller object. Use this constructor only, if you want to call simulations, that accept the controller object directly.

Parameters

<i>system</i>	Reference to the underlying dynamic system.
<i>plotter</i>	Pointer to a <a href="#">TebPlotter</a> object that enables visualization.

Definition at line 181 of file `simulator.h`.

16.30.3.3 `template<int p, int q> teb::Simulator< p, q >::~~Simulator ( ) [inline]`

Definition at line 184 of file `simulator.h`.

### 16.30.4 Member Function Documentation

16.30.4.1 `template<int p, int q> void teb::Simulator< p, q >::plotResults ( const std::vector< SimResults::TimeSeries > & time_series, teb::PlotOptions * options = nullptr ) const [protected]`

This method prepares a container of multiple `time_series` object in order to allow the utilization of the [TebPlotter](#) for visualization.

Customize the plot by optionally passing a [PlotOptions](#) object to this method.

Parameters

<i>time_series</i>	Container of time series.
<i>options</i>	Custom plot options.

Definition at line 390 of file `simulator.hpp`.

References `PRINT_INFO`.

16.30.4.2 `template<int p, int q> double teb::Simulator< p, q >::sampleTime ( ) [inline]`

See Also

[setSampleTime\(\)](#)

Returns

[Simulator](#) sample time.

Definition at line 203 of file simulator.h.

References `teb::Simulator< p, q >::_sample_time`.

16.30.4.3 `template<int p, int q> void teb::Simulator< p, q >::saturateControl ( Eigen::Ref< ControlVector > u )`  
[protected]

Parameters

<i>u</i>	The control input that should be bounded. The result will be stored directly to <i>u</i> .
----------	--

Definition at line 373 of file simulator.hpp.

16.30.4.4 `template<int p, int q> void teb::Simulator< p, q >::setControlInputSaturation ( const double * u_min, const double * u_max )` [inline]

Parameters

<i>u_min</i>	pointer to the minimum control bounds [q x 1]
<i>u_max</i>	pointer to the maximum control bounds [q x 1]

Definition at line 250 of file simulator.h.

References `teb::Simulator< p, q >::_control_bounds`.

16.30.4.5 `template<int p, int q> void teb::Simulator< p, q >::setIntegrator ( IntegratorPtr integrator )` [inline]

Parameters

<i>integrator</i>	Unique pointer to a <a href="#">NumericalIntegrator</a> subclass.
-------------------	---

Definition at line 215 of file simulator.h.

References `teb::Simulator< p, q >::_integrator`.

16.30.4.6 `template<int p, int q> void teb::Simulator< p, q >::setIntegrator ( NumericalIntegrators int_type )`

Parameters

<i>int_type</i>	Integrator type selected from <a href="#">teb::NumericalIntegrators</a> .
-----------------	---

Definition at line 352 of file simulator.hpp.

References `teb::EXPLICIT_EULER`, `teb::RUNGE_KUTTA_5TH`, and `teb::RUNGE_KUTTA_CLASSIC`.

16.30.4.7 `template<int p, int q> void teb::Simulator< p, q >::setPlotter ( TebPlotter * plotter )` [inline]



## Parameters

<i>plotter</i>	Pointer to the plotter object (no memory will be freed).
----------------	--

Definition at line 209 of file simulator.h.

References teb::Simulator< p, q >::\_plotter.

**16.30.4.8** `template<int p, int q> void teb::Simulator< p, q >::setPostStepCallback ( const std::function< void(BaseController *, SystemDynamics< p, q > *, Simulator< p, q > *)> & callback ) [inline]`

The callback function should have the following arguments:

- BaseController\* ctrl - Pointer to the current controller object
- SystemDynamics\* system - Pointer to the current system object
- Simulator\* sim - Pointer to the caller (TebSimulator class) itself

## Remarks

Use std::bind() in order to pass class methods

## Parameters

<i>callback</i>	function (object) that should be called
-----------------	---

Definition at line 279 of file simulator.h.

References teb::Simulator< p, q >::\_callback\_step\_post.

**16.30.4.9** `template<int p, int q> void teb::Simulator< p, q >::setPreSimCallback ( const std::function< void(BaseController *, SystemDynamics< p, q > *, Simulator< p, q > *)> & callback ) [inline]`

The callback function should have the following arguments:

- BaseController\* ctrl - Pointer to the current controller object
- SystemDynamics\* system - Pointer to the current system object
- Simulator\* sim - Pointer to the caller (TebSimulator class) itself

## Remarks

Use std::bind() in order to pass class methods

## Parameters

<i>callback</i>	function (object) that should be called
-----------------	---

Definition at line 294 of file simulator.h.

References teb::Simulator< p, q >::\_callback\_sim\_pre.

**16.30.4.10** `template<int p, int q> void teb::Simulator< p, q >::setPreStepCallback ( const std::function< void(BaseController *, SystemDynamics< p, q > *, Simulator< p, q > *)> & callback ) [inline]`

The callback function should have the following arguments:

- BaseController\* ctrl - Pointer to the current controller object
- SystemDynamics\* system - Pointer to the current system object
- Simulator\* sim - Pointer to the caller (TebSimulator class) itself

## Remarks

Use `std::bind()` in order to pass class methods

## Parameters

<i>callback</i>	function (object) that should be called
-----------------	---

Definition at line 265 of file `simulator.h`.

References `teb::Simulator< p, q >::_callback_step_pre`.

16.30.4.11 `template<int p, int q> void teb::Simulator< p, q >::setSampleTime ( double sample_time ) [inline]`

The system is simulated for the duration `sample_time` [sec] until a new control is applied. If `sample_time == -1`, the sample time is inherited from the [TebController](#).

**Todo** Allow different `sample_times` for controller and plant simulation (the TEB sample time  $\Delta T$  might be higher ...)

## Parameters

<i>sample_time</i>	Specified sample time.
--------------------	------------------------

Definition at line 196 of file `simulator.h`.

References `teb::Simulator< p, q >::_sample_time`.

16.30.4.12 `template<int p, int q> std::unique_ptr< SimResults > teb::Simulator< p, q >::simClosedLoop ( const double *const x0, const double *const xf, double sim_time, bool manual_stepping = false, bool plot_result = true, bool plot_steps = true, BaseController * ctrl = nullptr )`

This method performs a closed-loop simulation of the system controlled by the `TebSimulator`, both specified in the constructor of this class.

The TEB optimization is solved in each sampling-interval specified by [setSampleTime\(\)](#) to determine the next optimal control for the point to point transition from the most recent measurement of the state vector  $\mathbf{x}_k$  to the final state vector  $\mathbf{x}_f$ . As a starting point, the first state vector is given by  $\mathbf{x}_0$ .

**Bug** If we set a title or legend in the plot options for stepping (!), than the plot won't be rendered correctly.

## Parameters

<i>x0</i>	Double array with <i>p</i> values representing the start state vector
<i>xf</i>	Double array with <i>p</i> values representing the final state vector
<i>sim_time</i>	Total simulation time (duration of the simulation)
<i>manual_stepping</i>	if <code>true</code> , the simulation is paused after each sampling interval. During pause, the user can type in the desired number of steps until the next pause will be instantiated. If the user chooses "-1" steps, the simulation will be proceeded until <code>sim_time</code> is exceeded.
<i>plot_result</i>	if <code>true</code> , the results at the end of the simulation are plotted into a figure using <a href="#">TebPlotter</a> .
<i>plot_steps</i>	if <code>true</code> , the results after each <code>sample_inteval</code> are plotted into a figure using <a href="#">TebPlotter</a> (shows the progress of the control).
<i>ctrl</i>	Pointer to an external controller: set to <code>nullptr</code> in order to use the class member <code>_controller</code> .

## Returns

[SimResults](#) containing the time series of the closed-loop control.

Definition at line 119 of file `simulator.hpp`.

References `teb::SimResults::conservativeResize()`, `teb::BaseController::firstControl()`, `teb::BaseController::getDt()`, `INPUT_STREAM`, `teb::PlotOptions::legend`, `teb::PlotOptions::legend_entries`, `PRINT_INFO`, `PRINT_INFO_ONCE`, `teb::BaseController::resetController()`, `teb::SimResults::series`, `teb::PlotOptions::skip_last_value_right_column`, `teb::BaseController::step()`, and `teb::PlotOptions::title`.

**16.30.4.13** `template<int p, int q> void teb::Simulator< p, q >::simClosedLoop ( std::vector< std::pair< BaseController *, std::string >> & controllers, const double *const x0, const double *const xf, double sim_time, bool plot_result = true, bool plot_steps = true )`

This method performs an closed-loop simulation of the system controlled by different Controllers passed to this method.

The system is specified using the constructor of this class.

The TEB optimization is solved in each sampling-interval specified by `setSampleTime()` to determine the next optimal control for the point to point transition from the most recent measurement of the state vector  $\mathbf{x}_k$  to the final state vector  $\mathbf{x}_f$ . As a starting point, the first state vector is given by  $\mathbf{x}_0$ .

Parameters

<i>controllers</i>	array of different TebControllers with augmented name as string
<i>x0</i>	Double array with <i>p</i> values representing the start state vector
<i>xf</i>	Double array with <i>p</i> values representing the final state vector
<i>sim_time</i>	Total simulation time (duration of the simulation)
<i>plot_result</i>	if <code>true</code> , the results at the end of the simulation are plotted into a figure using <a href="#">TebPlotter</a> .
<i>plot_steps</i>	if <code>true</code> , the results after each sample_inteval are plotted into a figure using <a href="#">TebPlotter</a> (shows the progress of the control).

Definition at line 328 of file `simulator.hpp`.

References `teb::PlotOptions::legend`, `teb::PlotOptions::legend_entries`, `teb::SimResults::series`, `teb::PlotOptions::skip_last_value_right_column`, and `teb::PlotOptions::title`.

**16.30.4.14** `template<int p, int q> std::unique_ptr< SimResults > teb::Simulator< p, q >::simOpenAndClosedLoop ( const double *const x0, const double *const xf, double sim_time, bool manual_stepping = false, bool plot_steps = true )`

This method performs both, an open-loop simulation of the system controlled by the TebSimulator and the closed-loop simulation of the system.

Refer to `simOpenLoop()` and `simClosedLoop()` for details.

The results are combined and shown in a single figure using [TebPlotter](#).

Parameters

<i>x0</i>	Double array with <i>p</i> values representing the start state vector
<i>xf</i>	Double array with <i>p</i> values representing the final state vector
<i>sim_time</i>	Total simulation time for the closed-loop simulation (duration of the simulation)
<i>manual_stepping</i>	if <code>true</code> , the closed-loop simulation is paused after each sampling interval. During pause, the user can type in the desired number of steps until the next pause will be instantiated. If the user chooses "-1" steps, the simulation will be proceeded until <code>sim_time</code> is exceeded.
<i>plot_steps</i>	if <code>true</code> , the results after each sample_inteval of the closed-loop control are plotted into a figure using <a href="#">TebPlotter</a> (shows the progress of the control).

Returns

[SimResults](#) containing the time series of the open-loop control, the closed-loop control and the states and controls planned by the controller.

Definition at line 249 of file `simulator.hpp`.

References `teb::PlotOptions::legend`, `teb::PlotOptions::legend_entries`, `teb::SimResults::series`, `teb::PlotOptions::skip_last_value_right_column`, and `teb::PlotOptions::title`.

**16.30.4.15** `template<int p, int q> std::unique_ptr< SimResults > teb::Simulator< p, q >::simOpenLoop ( const double *const x0, const double *const xf, bool plot = true, BaseController * ctrl = nullptr )`

This method performs an open-loop simulation of the system controlled by the `TebSimulator`, both specified in the constructor of this class.

The TEB optimization is solved once to determine the optimal control for the point to point transition from the start state vector  $\mathbf{x}_0$  to the final state vector  $\mathbf{x}_f$ .

The determined control is applied to the simulated plant. In the absence of model mismatch and disturbances the output should be similar to the planned one by the TEB, if the system dynamics equations are satisfied.

#### Parameters

<i>x0</i>	Double array with <i>p</i> values representing the start state vector
<i>xf</i>	Double array with <i>p</i> values representing the final state vector
<i>plot</i>	if <code>true</code> , the results are plotted into a figure using <a href="#">TebPlotter</a> .
<i>ctrl</i>	Pointer to an external controller: set to <code>nullptr</code> in order to use the class member <code>_controller</code> .

#### Returns

[SimResults](#) containing the time series of the open-loop control and the states and controls planned by the controller.

Definition at line 28 of file `simulator.hpp`.

References `teb::SimResults::allocateMemory()`, `teb::BaseController::firstState()`, `teb::BaseController::getAbsoluteTimeVec()`, `teb::BaseController::getDt()`, `teb::BaseController::getN()`, `teb::BaseController::getStateCtrlInfoMat()`, `teb::PlotOptions::legend`, `teb::PlotOptions::legend_entries`, `teb::BaseController::resetController()`, `teb::BaseController::returnControlInputSequence()`, `teb::SimResults::series`, `teb::BaseController::step()`, and `teb::PlotOptions::title`.

**16.30.4.16** `template<int p, int q> void teb::Simulator< p, q >::simOpenLoop ( std::vector< std::pair< BaseController *, std::string >> & controllers, const double *const x0, const double *const xf, bool plot = true )`

This method performs an open-loop simulation of the system controlled by different Controllers passed to this method.

The system is specified using the constructor of this class.

The TEB optimization is solved once to determine the optimal control for the point to point transition from the start state vector  $\mathbf{x}_0$  to the final state vector  $\mathbf{x}_f$ .

#### Parameters

<i>controllers</i>	array of different <code>TebControllers</code> with augmented name as string
<i>x0</i>	Double array with <i>p</i> values representing the start state vector
<i>xf</i>	Double array with <i>p</i> values representing the final state vector
<i>plot</i>	if <code>true</code> , the results are plotted into a figure using <a href="#">TebPlotter</a> .

Definition at line 289 of file `simulator.hpp`.

References `teb::PlotOptions::legend`, `teb::PlotOptions::legend_entries`, `teb::SimResults::series`, `teb::PlotOptions::skip_last_value_right_column`, and `teb::PlotOptions::title`.

16.30.4.17 `template<int p, int q> StateVector teb::Simulator< p, q >::systemStep ( const Eigen::Ref< const StateVector > & x_k, const Eigen::Ref< const ControlVector > & u_k, double dt ) [inline]`

Definition at line 238 of file simulator.h.

References `teb::Simulator< p, q >::_integrator`, and `teb::Simulator< p, q >::_system`.

## 16.30.5 Friends And Related Function Documentation

16.30.5.1 `template<int p, int q> friend class SimResults [friend]`

Definition at line 150 of file simulator.h.

16.30.5.2 `template<int p, int q> template<int pf, int qf> friend class SystemDynamics [friend]`

Definition at line 148 of file simulator.h.

## 16.30.6 Member Data Documentation

16.30.6.1 `template<int p, int q> std::function<void(BaseController*,SystemDynamics<p,q>*, Simulator*)> teb::Simulator< p, q >::_callback_sim_pre [protected]`

Definition at line 334 of file simulator.h.

Referenced by `teb::Simulator< p, q >::setPreSimCallback()`.

16.30.6.2 `template<int p, int q> std::function<void(BaseController*,SystemDynamics<p,q>*, Simulator*)> teb::Simulator< p, q >::_callback_step_post [protected]`

Definition at line 329 of file simulator.h.

Referenced by `teb::Simulator< p, q >::setPostStepCallback()`.

16.30.6.3 `template<int p, int q> std::function<void(BaseController*,SystemDynamics<p,q>*, Simulator*)> teb::Simulator< p, q >::_callback_step_pre [protected]`

Definition at line 324 of file simulator.h.

Referenced by `teb::Simulator< p, q >::setPreStepCallback()`.

16.30.6.4 `template<int p, int q> std::pair<ControlVector,ControlVector> teb::Simulator< p, q >::_control_bounds = std::make_pair(ControlVector::Constant(-INF), ControlVector::Constant(INF)) [protected]`

Definition at line 318 of file simulator.h.

Referenced by `teb::Simulator< p, q >::setControlInputSaturation()`.

16.30.6.5 `template<int p, int q> BaseController* teb::Simulator< p, q >::_controller = nullptr [protected]`

Definition at line 310 of file simulator.h.

16.30.6.6 `template<int p, int q> IntegratorPtr teb::Simulator< p, q >::_integrator = IntegratorPtr(new RungeKutta5thOrder<p,q>())` `[protected]`

Definition at line 312 of file simulator.h.

Referenced by `teb::Simulator< p, q >::setIntegrator()`, and `teb::Simulator< p, q >::systemStep()`.

16.30.6.7 `template<int p, int q> TebPlotter* teb::Simulator< p, q >::_plotter = nullptr` `[protected]`

Definition at line 314 of file simulator.h.

Referenced by `teb::Simulator< p, q >::setPlotter()`.

16.30.6.8 `template<int p, int q> double teb::Simulator< p, q >::_sample_time = -1.` `[protected]`

Definition at line 315 of file simulator.h.

Referenced by `teb::Simulator< p, q >::sampleTime()`, and `teb::Simulator< p, q >::setSampleTime()`.

16.30.6.9 `template<int p, int q> SystemDynamics<p,q>* teb::Simulator< p, q >::_system = nullptr` `[protected]`

Definition at line 311 of file simulator.h.

Referenced by `teb::Simulator< p, q >::systemStep()`.

The documentation for this class was generated from the following files:

- [simulator.h](#)
- [simulator.hpp](#)

## 16.31 teb::Config::Optim::Solver Struct Reference

Configurations related to solvers.

```
#include <config.h>
```

### Classes

- struct [Lsq](#)  
*Configurations especially for least-squares-solvers.*
- struct [Nlopt](#)  
*Configurations especially for the nlopt solver wrapper.*
- struct [NonlinearProgram](#)  
*Settings for constrained optimization solver (nonlinear program solver)*

### Public Attributes

- unsigned int [solver\\_iter](#) = 5
- struct [teb::Config::Optim::Solver::Lsq](#) [lsq](#)  
*Configurations especially for least-squares-solvers.*
- struct [teb::Config::Optim::Solver::NonlinearProgram](#) [nonlin\\_prog](#)

*Settings for constrained optimization solver (nonlinear program solver)*

- struct [teb::Config::Optim::Solver::Nlopt](#) `nlopt`

### 16.31.1 Detailed Description

Definition at line 78 of file `config.h`.

### 16.31.2 Member Data Documentation

#### 16.31.2.1 struct `teb::Config::Optim::Solver::Lsq` `teb::Config::Optim::Solver::lsq`

Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [rocket\\_system.cpp](#).

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, and `teb::BaseSolverLeastSquares::buildValueVector()`.

#### 16.31.2.2 struct `teb::Config::Optim::Solver::Nlopt` `teb::Config::Optim::Solver::nlopt`

#### 16.31.2.3 struct `teb::Config::Optim::Solver::NonlinearProgram` `teb::Config::Optim::Solver::nonlin_prog`

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.31.2.4 unsigned int `teb::Config::Optim::Solver::solver_iter` = 5

Definition at line 80 of file `config.h`.

The documentation for this struct was generated from the following file:

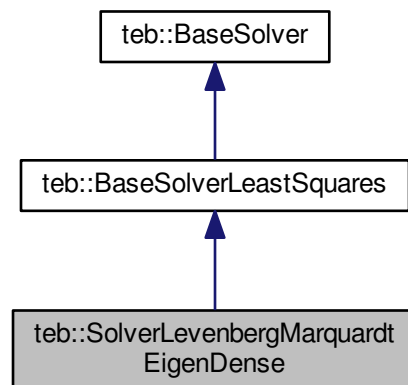
- [config.h](#)

## 16.32 `teb::SolverLevenbergMarquardtEigenDense` Class Reference

Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).

```
#include <solver_levenbergmarquardt_eigen_dense.h>
```

Inheritance diagram for `teb::SolverLevenbergMarquardtEigenDense`:



## Public Types

- using `VertexContainer` = `HyperGraph::VertexContainer`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using `EdgeContainer` = `HyperGraph::EdgeContainer`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

## Public Member Functions

- `SolverLevenbergMarquardtEigenDense` ()  
*Empty Constructor.*
- void `setConfig` (const `Config` \*config)  
*Set config including solver settings.*
- virtual bool `solve` (`HyperGraph` \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

## Public Attributes

- const `Config` \* `cfg` = nullptr  
*Store pointer to `Config` object.*
- `EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

## Protected Member Functions

- virtual bool `solveImpl` ()  
*Solve the actual optimization problem.*
- void `buildJacobian` ()  
*Construct the jacobian of the complete least-squares optimization problem.*
- virtual void `initWorkspaces` ()  
*Initialize workspaces for the block jacobians and hessians.*



- void [buildValueVector](#) ()  
*Build value / cost vector including all objectives and constraints.*
- int [getValueDimension](#) () const  
*Get dimension of the composed value/cost vector (including objectives and constraints).*
- double [getChi2](#) () const  
*Calculate the  $\chi^2$  error of the optimization problem.*
- void [adaptWeights](#) ()  
*Automatic weight adaptation that increases soft constraint weights after each outer teb iteration.*
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const  
*Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- Eigen::MatrixXd [\\_jacobian](#)  
*Store the jacobian matrix of the complete least-squares optimization problem.*
- Eigen::VectorXd [\\_values](#)  
*Store the value vector computed in [buildValueVector\(\)](#).*
- int [\\_weight\\_adapt\\_count](#) = 0  
*Store current state of the weight adaptation method [adaptWeights\(\)](#).*
- double [\\_weight\\_equalities](#) = 1  
*Store current weight for equality soft-constraints.*
- double [\\_weight\\_inequalities](#) = 1  
*Store current weight for inequality soft-constraints.*
- int [\\_val\\_dim](#) = -1  
*Store dimension of the value vector here (see [getValueDimension\(\)](#)).*
- [VertexContainer](#) \* [\\_active\\_vertices](#) = nullptr  
*Pointer to active vertex container (active = non-fixed)*
- [EdgeContainer](#) \* [\\_objectives](#) = nullptr  
*Pointer to edges representing objective functions.*
- [EdgeContainer](#) \* [\\_equalities](#) = nullptr

- Pointer to edges representing equality constraints.*
- `EdgeContainer * _inequalities = nullptr`  
*Pointer to edges representing inequality constraints.*
- `int _opt_vec_dim = -1`  
*Store dimension of the optimization vector here (see [getOptVecDimension\(\)](#)).*
- `int _objective_dim = -1`  
*Store dimension of the objective function (see [getDimObjectives\(\)](#)).*
- `int _equalities_dim = -1`  
*Store dimension of the equality constraints (see [getDimEqualities\(\)](#)).*
- `int _inequalities_dim = -1`  
*Store dimension of the inequality constraints (see [getDimInequalities\(\)](#)).*
- `unsigned int _no_vert_backups = 0`  
*Track number of vertices backups made.*
- `bool _graph_structure_modified = true`  
*Mark if a new graph structure is available, thus we need to reinitialize all workspaces.*

### 16.32.1 Detailed Description

This solver implements the Levenberg-Marquardt algorithm also implemented in [2]. The underlying linear system is solved using Eigens Dense Matrix algebra ([http://eigen.tuxfamily.org/dox/group\\_\\_Tutorial-LinearAlgebra.html](http://eigen.tuxfamily.org/dox/group__Tutorial-LinearAlgebra.html)).

The jacobian required for the optimization routine is stored in a dense matrix. Therefore no exploitation of the sparse structure for solving the linear system is taken into account.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

Definition at line 25 of file `solver_levenbergmarquardt_eigen_dense.h`.

### 16.32.2 Member Typedef Documentation

16.32.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer` `[inherited]`

Definition at line 36 of file `base_solver.h`.

16.32.2.2 `using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer` `[inherited]`

Definition at line 34 of file `base_solver.h`.

### 16.32.3 Constructor & Destructor Documentation

16.32.3.1 `teb::SolverLevenbergMarquardtEigenDense::SolverLevenbergMarquardtEigenDense ( )` `[inline]`

Definition at line 30 of file `solver_levenbergmarquardt_eigen_dense.h`.

### 16.32.4 Member Function Documentation

16.32.4.1 `void teb::BaseSolverLeastSquares::adaptWeights ( )` `[protected]`, `[inherited]`

This method increases the soft constraint weights for inequalities and equalities after each outer TEB optimization loop (see [TebController::optimizeTEB\(\)](#)).

After the outer TEB loop is completed ([Config::Teb::teb\\_iter](#)), the weights are set back to [Config::Optim::Solver::Lsq::weight\\_equalities](#) and [Config::Optim::Solver::Lsq::weight\\_inequalities](#).

The weights are increased by  $\sigma = \sigma_{init} \cdot \gamma^i$ .  $\gamma$  denotes the increasement factor [Config::Optim::Solver::Lsq::weight\\_adaptation\\_factor](#).  $i$  denotes the current iteration number.  $\sigma_{init}$  is obtained from the config (see above).

Call this method at the beginning of [solveImpl\(\)](#)!

#### Remarks

This procedere forces the optimization result to first contract the trajectory in sense of the objective, and afterwards trying to satisfy the constraints. Starting with very high weights in advance could lead to abrupt gradients for the solver. In addition the effect for the objectives (e.g. time optimallity) could be slow, if the corresponding gradients are extremly small in comparision to constraint gradients.

#### See Also

[TebController::optimizeTEB\(\)](#)

Definition at line 100 of file `base_solver_least_squares.cpp`.

References [teb::BaseSolverLeastSquares::\\_weight\\_adapt\\_count](#), [teb::BaseSolverLeastSquares::\\_weight\\_equalities](#), [teb::BaseSolverLeastSquares::\\_weight\\_inequalities](#), [teb::BaseSolver::cfg](#), [teb::Config::Optim::Solver::lsq](#), [teb::Config::optim](#), [teb::Config::Optim::solver](#), [teb::Config::teb](#), [teb::Config::Teb::teb\\_iter](#), [teb::Config::Optim::Solver::Lsq::weight\\_adaptation\\_factor](#), [teb::Config::Optim::Solver::Lsq::weight\\_equalities](#), and [teb::Config::Optim::Solver::Lsq::weight\\_inequalities](#).

Referenced by [solveImpl\(\)](#), and [teb::SolverLevenbergMarquardtEigenSparse::solveImpl\(\)](#).

**16.32.4.2** `void teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::plusFree\(\)](#) for each vertex.

#### Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length euquals the dimension of all free variables.
--------------	--

Definition at line 127 of file `base_solver.h`.

References [teb::BaseSolver::\\_active\\_vertices](#).

Referenced by [solveImpl\(\)](#), [teb::SolverLevenbergMarquardtEigenSparse::solveImpl\(\)](#), and [teb::SolverSQPDense::solveImpl\(\)](#).

**16.32.4.3** `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::setFree\(\)](#) for each vertex.

#### Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file `base_solver.h`.

References [teb::BaseSolver::\\_active\\_vertices](#).

**16.32.4.4** `void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]`

See Also

[VertexType::push\(\)](#)

Definition at line 223 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.32.4.5 void teb::SolverLevenbergMarquardtEigenDense::buildJacobian ( ) [protected]

This method constructs the jacobian of the complete optimization problem.

The size of the jacobian is `[getValueDimension() x getOptVecDimension()]`. This method queries the `EdgeType::computeJacobian()` of each edge. The column index for each entry is determined using `TebVertex::getOptVecIdx` and the number of free dimensions for each vertex.

Since this solver relies on soft-constraints for equality and inequality constraints, the chain-rule for calculating derivatives is utilized. Equality constraints are treated as usual objective, therefore the jacobian does not need to be changed.

For inequality constraints, the derivative of  $\max(c(x), 0)$  is

$$\begin{cases} \frac{\partial}{\partial x} c_i(x) & \text{if } c_i(x) > 0 \\ 0 & \text{otherwise} \end{cases} = \mathbf{J}_{ineq}(:, j = 1 : n) .* \frac{1}{\mathbf{c}(x)} .* \max(\mathbf{c}(x), 0)$$

The max operator is applied component-wise. '.\*' corresponds to Matlab's component-wise multiplication.

The resulting jacobian matrix is stored to `SolverLevenbergMarquardtEigenDense::_jacobian`;

Definition at line 157 of file `solver_levenbergmarquardt_eigen_dense.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, `_jacobian`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::BaseSolverLeastSquares::_val_dim`, `teb::BaseSolverLeastSquares::_weight_equalities`, `teb::BaseSolverLeastSquares::_weight_inequalities`, `teb::BaseSolver::cfg`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, `teb::VertexType::isFixedComp()`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::Solver::Lsq::soft_constr_epsilon`, and `teb::Config::Optim::solver`.

Referenced by `solveImpl()`.

#### 16.32.4.6 void teb::BaseSolverLeastSquares::buildValueVector ( ) [protected],[inherited]

This method constructs the full value/cost vector  $\mathbf{f}$  (not  $\mathbf{f}^2$ ) for the underlying least-squares optimization problem.

The length of the vector can be obtained using `getValueDimension()`.

Constraints are transformed into costs/objectives using soft constraints:

- Equality constraints  $c(x) = 0$ .

These constraints are directly taken as objectives since  $c^2(x)$  has a local minima at  $x = 0$ .

- Inequality constraints  $c(x) \leq 0$

These constraints are transformed using  $f = \max(c(x), 0)$  (component-wise).

This notation is similar to: `c(x) <= epsilon ? 0 : c(x)`.

Afterwards (in the actual `solveImpl()` method, the cost function will be squared, that leads to twice differentiable costs for the constraints, if  $f$  is piecewise differentiable once and if it intersects with the abscissa.

In addition the weights `BaseSolverLeastSquares::_weight_equalities` and `BaseSolverLeastSquares::_weight_inequalities` are taken into account in order to weight the "soft constraint objectives". To make the weights

comparable to [2] and our Matlab TEB version, we take the square root of both weights, since the least-square problem here is formulated as  $f^2(x, \sigma)$  instead of  $\sigma f^2(x)$ .  $\sigma$  denotes the weight.

The results are stored internally to `BaseSolverLeastSquares::_values`.

See Also

[adaptWeights\(\)](#), [getValueDimension\(\)](#)

Definition at line 31 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `teb::BaseSolverLeastSquares::_val_dim`, `teb::BaseSolverLeastSquares::_values`, `teb::BaseSolverLeastSquares::_weight_equalities`, `teb::BaseSolverLeastSquares::_weight_inequalities`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::Solver::Lsq::soft_constr_epsilon`, and `teb::Config::Optim::solver`.

Referenced by `solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.32.4.7** `void teb::BaseSolver::discardBackupVertices ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::BaseSolver::solve()`, `solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.32.4.8** `double teb::BaseSolverLeastSquares::getChi2 ( )` `const` `[inline]`, `[protected]`, `[inherited]`

The  $\chi^2$  error is the scalar cost value of the least-squares optimization problem since the total cost function is composed of weighted terms:

$$f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \mathbf{f}(\mathcal{B})$$

Weights are included in  $\mathbf{f}(\mathcal{B})$ . In this case it is  $\chi^2 = f(\mathcal{B})$ .

**Todo** Maybe switch to the formulation  $f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \mathbf{\Omega} \mathbf{f}(\mathcal{B})$  similar to g2o and Teb-Matlab.

Definition at line 82 of file `base_solver_least_squares.h`.

References `teb::BaseSolverLeastSquares::_values`.

Referenced by `solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.32.4.9** `int teb::BaseSolver::getDimEqualities ( )` `const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file `base_solver.h`.

References `teb::BaseSolver::_equalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.32.4.10** `int teb::BaseSolver::getDimInequalities ( )` `const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file `base_solver.h`.

References `teb::BaseSolver::_inequalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.32.4.11** `int teb::BaseSolver::getDimObjectives ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file `base_solver.h`.

References `teb::BaseSolver::_objectives`.

Referenced by `teb::BaseSolver::solve()`.

**16.32.4.12** `Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const` `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls `VertexType::getDataFree()` for each vertex.

#### Remarks

Make sure `BaseSolver::_opt_vec_dim` is valid (see `solve()`).

#### Returns

`Eigen::Vector` containing all values. The length equals the dimension of all free variables (`getOptVecDimension()`).

Definition at line 162 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_opt_vec_dim`.

**16.32.4.13** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

#### See Also

`TebController::getActiveVertices()`, `getValueDimension()`

Definition at line 180 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

**16.32.4.14** `int teb::BaseSolverLeastSquares::getValueDimension ( ) const` `[protected]`, `[inherited]`

This method returns the dimension of the full cost/value vector including all objective and constraints.

In particular it sums up all dimensions of single edges stored in the given hyper-graph.

#### Remarks

Make sure `BaseSolver::_objective_dim`, `BaseSolver::_equalities_dim` and `BaseSolver::_inequalities_dim` are valid (see `solve()`).

#### Returns

dimension of the complete value/cost vector.

## See Also

[buildValueVector\(\)](#), [getOptVecDimension\(\)](#), [getDimObjectives\(\)](#), [getDimEqualities\(\)](#), [getDimInequalities\(\)](#)

Definition at line 73 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_inequalities_dim`, and `teb::BaseSolver::_objective_dim`.

Referenced by `solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.32.4.15** `virtual void teb::BaseSolverLeastSquares::initWorkspaces ( )` `[inline]`, `[protected]`, `[virtual]`, `[inherited]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each [EdgeType](#)) or map to existing memory.

Reimplemented from [teb::BaseSolver](#).

Definition at line 43 of file `base_solver_least_squares.h`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `PRINT_DEBUG_COND_ONCE`, and `teb::QUADRATIC`.

**16.32.4.16** `void teb::BaseSolver::restoreVertices ( )` `[inline]`, `[protected]`, `[inherited]`

## See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.32.4.17** `void teb::BaseSolver::restoreVerticesButKeepBackup ( )` `[inline]`, `[protected]`, `[inherited]`

## See Also

[VertexType::top\(\)](#)

Definition at line 227 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.32.4.18** `void teb::BaseSolver::setConfig ( const Config * config )` `[inline]`, `[inherited]`

## Remarks

This function is called from the TEB class within `setSolver` method.

## Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file `base_solver.h`.

References `teb::BaseSolver::cfg`.

Referenced by `teb::TebController< p, q >::setSolver()`.

**16.32.4.19** `virtual bool teb::BaseSolver::solve ( HyperGraph * optimizable_graph ) [inline],[virtual],[inherited]`

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with [solveImpl\(\)](#).

**Todo** Split initWorkspaces into init and update phase in order to hot-start from previous initializations.

#### Parameters

<i>optimizable_graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
--------------------------	---

Definition at line 61 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolver::_no_vert_backups`, `teb::BaseSolver::_objective_dim`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::HyperGraph::activeVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::HyperGraph::equalities()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolver::getOptVecDimension()`, `teb::HyperGraph::inequalities()`, `teb::BaseSolver::initWorkspaces()`, `teb::HyperGraph::isGraphModified()`, `teb::HyperGraph::objectives()`, and `teb::BaseSolver::solveImpl()`.

**16.32.4.20** `bool teb::SolverLevenbergMarquardtEigenDense::solveImpl ( ) [protected],[virtual]`

Store results to the vertices of the active vertices container.

#### Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolverLeastSquares](#).

Definition at line 8 of file `solver_levenbergmarquardt_eigen_dense.cpp`.

References `_jacobian`, `teb::BaseSolver::_opt_vec_dim`, `teb::BaseSolverLeastSquares::_val_dim`, `teb::BaseSolverLeastSquares::_values`, `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::backupVertices()`, `buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolverLeastSquares::getChi2()`, `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::restoreVertices()`.

## 16.32.5 Member Data Documentation

**16.32.5.1** `VertexContainer* teb::BaseSolver::_active_vertices = nullptr [protected],[inherited]`

Definition at line 233 of file `base_solver.h`.

Referenced by `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::applyOptVec()`, `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::getOptVecCopy()`, `teb::BaseSolver::getOptVecDimension()`, `teb::BaseSolver::restoreVertices()`, `teb::BaseSolver::restoreVerticesButKeepBackup()`, and `teb::BaseSolver::solve()`.

**16.32.5.2** `EdgeContainer* teb::BaseSolver::_equalities = nullptr [protected],[inherited]`

Definition at line 235 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `buildJacobian()`, `teb::SolverLevenbergMarquardt-`



`EigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

#### 16.32.5.3 `int teb::BaseSolver::_equalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 240 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverNloptPackage::getEqConstrDimFromStorage()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.32.5.4 `bool teb::BaseSolver::_graph_structure_modified = true` `[protected]`, `[inherited]`

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the `solve()` method.

Definition at line 250 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.32.5.5 `EdgeContainer* teb::BaseSolver::_inequalities = nullptr` `[protected]`, `[inherited]`

Definition at line 236 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsInEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

#### 16.32.5.6 `int teb::BaseSolver::_inequalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 241 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.32.5.7 `Eigen::MatrixXd teb::SolverLevenbergMarquardtEigenDense::_jacobian` `[protected]`

Definition at line 40 of file `solver_levenbergmarquardt_eigen_dense.h`.

Referenced by `buildJacobian()`, and `solveImpl()`.

**16.32.5.8** `unsigned int teb::BaseSolver::_no_vert_backups = 0` [protected],[inherited]

Definition at line 243 of file `base_solver.h`.

Referenced by `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::restoreVertices()`, and `teb::BaseSolver::solve()`.

**16.32.5.9** `int teb::BaseSolver::_objective_dim = -1` [protected],[inherited]

Definition at line 239 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::solve()`.

**16.32.5.10** `EdgeContainer* teb::BaseSolver::_objectives = nullptr` [protected],[inherited]

Definition at line 234 of file `base_solver.h`.

Referenced by `buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNz()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::SolverNloptPackage::objectives()`, and `teb::BaseSolver::solve()`.

**16.32.5.11** `int teb::BaseSolver::_opt_vec_dim = -1` [protected],[inherited]

Definition at line 238 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNz()`, `teb::BaseSolver::getOptVecCopy()`, `teb::SolverNloptPackage::getOptVecDimFromStorage()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, `solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.32.5.12** `int teb::BaseSolverLeastSquares::_val_dim = -1` [protected],[inherited]

Definition at line 96 of file `base_solver_least_squares.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.32.5.13** `Eigen::VectorXd teb::BaseSolverLeastSquares::_values` [protected],[inherited]

Definition at line 90 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverLeastSquares::getChi2()`, `solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.32.5.14** `int teb::BaseSolverLeastSquares::_weight_adapt_count = 0` [protected],[inherited]

Definition at line 92 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`.

**16.32.5.15** `double teb::BaseSolverLeastSquares::_weight_equalities = 1` `[protected]`, `[inherited]`

Definition at line 93 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, and `teb::BaseSolverLeastSquares::buildValueVector()`.

**16.32.5.16** `double teb::BaseSolverLeastSquares::_weight_inequalities = 1` `[protected]`, `[inherited]`

Definition at line 94 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, and `teb::BaseSolverLeastSquares::buildValueVector()`.

**16.32.5.17** `const Config* teb::BaseSolver::cfg = nullptr` `[inherited]`

See Also

[setConfig\(\)](#)

Definition at line 95 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::setConfig()`, and `teb::SolverSQPDense::solveImpl()`.

**16.32.5.18** `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW` `[inherited]`

Definition at line 253 of file `base_solver.h`.

The documentation for this class was generated from the following files:

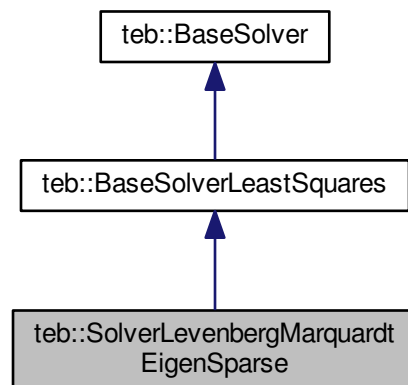
- [solver\\_levenbergmarquardt\\_eigen\\_dense.h](#)
- [solver\\_levenbergmarquardt\\_eigen\\_dense.cpp](#)

## 16.33 `teb::SolverLevenbergMarquardtEigenSparse` Class Reference

Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).

```
#include <solver_levenbergmarquardt_eigen_sparse.h>
```

Inheritance diagram for `teb::SolverLevenbergMarquardtEigenSparse`:



## Public Types

- using `VertexContainer` = `HyperGraph::VertexContainer`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using `EdgeContainer` = `HyperGraph::EdgeContainer`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

## Public Member Functions

- `SolverLevenbergMarquardtEigenSparse ()`  
*Empty Constructor.*
- void `setConfig` (const `Config` \*config)  
*Set config including solver settings.*
- virtual bool `solve` (`HyperGraph` \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

## Public Attributes

- const `Config` \* `cfg` = nullptr  
*Store pointer to `Config` object.*
- `EIGEN_MAKE_ALIGNED_OPERATOR_NEW`

## Protected Member Functions

- virtual bool `solveImpl` ()  
*Solve the actual optimization problem.*
- void `allocateSparseJacobian` ()  
*Allocate sparse jacobian.*
- void `buildJacobian` ()  
*Construct the jacobian of the complete least-squares optimization problem.*

- void [countJacobianColNNZ](#) ()  
*Count number of non-zeros per column in order to initialize jacobain sparse matrix.*
- virtual void [initWorkspaces](#) ()  
*Initialize workspaces for the block jacobians and hessians.*
- void [buildValueVector](#) ()  
*Build value / cost vector including all objectives and constraints.*
- int [getValueDimension](#) () const  
*Get dimension of the composed value/cost vector (including objectives and constraints).*
- double [getChi2](#) () const  
*Calculate the  $\chi^2$  error of the optimization problem.*
- void [adaptWeights](#) ()  
*Automatic weight adaptation that increases soft constraint weights after each outer teb iteration.*
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const  
*Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- Eigen::SparseMatrix< double > [\\_jacobian](#)  
*Store the jacobian matrix of the complete least-squares optimziation problem.*
- Eigen::VectorXi [\\_nnz\\_per\\_col](#)  
*Store number of non-zeros per column of the jacobian matrix.*
- Eigen::SimplicialLDLT  
< Eigen::SparseMatrix< double >  
, Eigen::Upper > [\\_sparse\\_solver](#)  
*Eigens sparse solver wrapper.*
- Eigen::VectorXd [\\_values](#)  
*Store the value vector computed in [buildValueVector\(\)](#).*
- int [\\_weight\\_adapt\\_count](#) = 0  
*Store current state of the weight adaptation method [adaptWeights\(\)](#).*
- double [\\_weight\\_equalities](#) = 1  
*Store current weight for equality soft-constraints.*

- double `_weight_inequalities` = 1  
*Store current weight for inequality soft-constraints.*
- int `_val_dim` = -1  
*Store dimension of the value vector here (see [getValueDimension\(\)](#)).*
- `VertexContainer` \* `_active_vertices` = nullptr  
*Pointer to active vertex container (active = non-fixed)*
- `EdgeContainer` \* `_objectives` = nullptr  
*Pointer to edges representing objective functions.*
- `EdgeContainer` \* `_equalities` = nullptr  
*Pointer to edges representing equality constraints.*
- `EdgeContainer` \* `_inequalities` = nullptr  
*Pointer to edges representing inequality constraints.*
- int `_opt_vec_dim` = -1  
*Store dimension of the optimization vector here (see [getOptVecDimension\(\)](#)).*
- int `_objective_dim` = -1  
*Store dimension of the objective function (see [getDimObjectives\(\)](#)).*
- int `_equalities_dim` = -1  
*Store dimension of the equality constraints (see [getDimEqualities\(\)](#)).*
- int `_inequalities_dim` = -1  
*Store dimension of the inequality constraints (see [getDimInequalities\(\)](#)).*
- unsigned int `_no_vert_backups` = 0  
*Track number of vertices backups made.*
- bool `_graph_structure_modified` = true  
*Mark if a new graph structure is available, thus we need to reinitialize all workspaces.*

### 16.33.1 Detailed Description

This solver implements the Levenberg-Marquardt algorithm also implemented in [2]. The underlying linear system is solved using Eigens Sparse Matrix algebra ( [http://eigen.tuxfamily.org/dox/group\\_\\_Topic-SparseSystems.html](http://eigen.tuxfamily.org/dox/group__Topic-SparseSystems.html) ).

The jacobian required for the optimization routine is stored in a sparse matrix.

**Todo** Consider only the upper diagonal part of the hessian?

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### Examples:

[integrator\\_system1.cpp](#), [integrator\\_system2.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [integrator\\_system\\_mex.cpp](#), [integrator\\_system\\_sfun.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [linear\\_system\\_state\\_space.cpp](#), [mobile\\_robot\\_teb.cpp](#), [rocket\\_system.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Definition at line 31 of file `solver_levenbergmarquardt_eigen_sparse.h`.

### 16.33.2 Member Typedef Documentation

16.33.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer` [inherited]

Definition at line 36 of file `base_solver.h`.

16.33.2.2 using `teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer` `[inherited]`

Definition at line 34 of file `base_solver.h`.

### 16.33.3 Constructor & Destructor Documentation

16.33.3.1 `teb::SolverLevenbergMarquardtEigenSparse::SolverLevenbergMarquardtEigenSparse ( )` `[inline]`

Definition at line 36 of file `solver_levenbergmarquardt_eigen_sparse.h`.

### 16.33.4 Member Function Documentation

16.33.4.1 void `teb::BaseSolverLeastSquares::adaptWeights ( )` `[protected]`, `[inherited]`

This method increases the soft constraint weights for inequalities and equalities after each outer TEB optimization loop (see [TebController::optimizeTEB\(\)](#)).

After the outer TEB loop is completed (`Config::Teb::teb_iter`), the weights are set back to `Config::Optim::Solver::Lsq::weight_equalities` and `Config::Optim::Solver::Lsq::weight_inequalities`.

The weights are increased by  $\sigma = \sigma_{init} \cdot \gamma^i$ .  $\gamma$  denotes the increasement factor `Config::Optim::Solver::Lsq::weight_adaptation_factor`.  $i$  denotes the current iteration number.  $\sigma_{init}$  is obtained from the config (see above).

Call this method at the beginning of `solveImpl()`!

#### Remarks

This procedere forces the optimization result to first contract the trajectory in sense of the objective, and afterwards trying to satisfy the constraints. Starting with very high weights in advance could lead to abrupt gradients for the solver. In addition the effect for the objectives (e.g. time optimallity) could be slow, if the corresponding gradients are extremely small in comparision to constraint gradients.

#### See Also

[TebController::optimizeTEB\(\)](#)

Definition at line 100 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolverLeastSquares::_weight_adapt_count`, `teb::BaseSolverLeastSquares::_weight_equalities`, `teb::BaseSolverLeastSquares::_weight_inequalities`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::solver`, `teb::Config::teb`, `teb::Config::Teb::teb_iter`, `teb::Config::Optim::Solver::Lsq::weight_adaptation_factor`, `teb::Config::Optim::Solver::Lsq::weight_equalities`, and `teb::Config::Optim::Solver::Lsq::weight_inequalities`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

16.33.4.2 void `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian ( )` `[inline]`, `[protected]`

Definition at line 44 of file `solver_levenbergmarquardt_eigen_sparse.h`.

References `_jacobian`, `_nnz_per_col`, `teb::BaseSolver::_opt_vec_dim`, `teb::BaseSolverLeastSquares::_val_dim`, and `countJacobianColINNZ()`.

Referenced by `buildJacobian()`.

16.33.4.3 void `teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta )` `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls [VertexType::plusFree\(\)](#) for each vertex.

## Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length equals the dimension of all free variables.
--------------	---

Definition at line 127 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.33.4.4** `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls `VertexType::setFree()` for each vertex.

## Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`.

**16.33.4.5** `void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]`

## See Also

[VertexType::push\(\)](#)

Definition at line 223 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.33.4.6** `void teb::SolverLevenbergMarquardtEigenSparse::buildJacobian ( ) [protected]`

This method constructs the jacobian of the complete optimization problem.

The size of the jacobian is `[getValueDimension() x getOptVecDimension()]`. This method queries the `EdgeType::computeJacobian()` of each edge. The column index for each entry is determined using `TebVertex::getOptVecIdx` and the number of free dimensions for each vertex.

Since this solver relies on soft-constraints for equality and inequality constraints, the chain-rule for calculating derivatives is utilized. Equality constraints are treated as usual objective, therefore the jacobian does not need to be changed.

For inequality constraints, the derivative of  $\max(c(x), 0)$  is

$$\begin{cases} \frac{\partial}{\partial x} c_i(x) & \text{if } c_i(x) > 0 \\ 0 & \text{otherwise} \end{cases} = \mathbf{J}_{ineq}(:, j = 1 : n) \cdot \frac{1}{\mathbf{c}(x)} \cdot \max(\mathbf{c}(x), \mathbf{0})$$

The max operator is applicated component-wise. `' .* '` corresponds to Matlab's component-wise multiplication.

The resulting jacobian matrix is stored to `SolverLevenbergMarquardtEigenSparse::_jacobian`;

**Todo** Check if implemented ordering of column and row iterations fits Eigen's column major representation best.

Definition at line 162 of file solver\_levenbergmarquardt\_eigen\_sparse.cpp.



References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, `_jacobian`, `teb::BaseSolver::_objectives`, `teb::BaseSolverLeastSquares::_weight_equalities`, `teb::BaseSolverLeastSquares::_weight_inequalities`, `allocateSparseJacobian()`, `teb::BaseSolver::cfg`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, `teb::VertexType::isFixedComp()`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::Solver::Lsq::soft_constr_epsilon`, and `teb::Config::Optim::solver`.

Referenced by `solveImpl()`.

#### 16.33.4.7 void teb::BaseSolverLeastSquares::buildValueVector ( ) [protected], [inherited]

This method constructs the full value/cost vector  $\mathbf{f}$  (not  $\mathbf{f}^2$ ) for the underlying least-squares optimization problem.

The lenght of the vector can be obtained using `getValueDimension()`.

Constraints are transformed into costs/objectives using soft constraints:

- Equality constraints  $c(x) = 0$ .  
These constraints are directly taken as objectives since  $c^2(x)$  has a local minima at  $x = 0$ .
- Inequality constraints  $c(x) \leq 0$   
These constraints are transformed using  $f = \max(c(x), 0)$  (component-wise).  
This notation is similar to:  $c(x) \leq \epsilon \ ? \ 0 : c(x)$ .  
Afterwards (in the actual `solveImpl()` method, the cost function will be squared, that leads to twice differentiable costs for the constraints, if  $f$  is piecewise differentiable once and if it intersects with the abscissa.

In addition the weights `BaseSolverLeastSquares::_weight_equalities` and `BaseSolverLeastSquares::_weight_inequalities` are taken into account in order to weight the "soft constraint objectives". To make the weights comparable to [2] and our Matlab TEB version, we take the square root of both weights, since the least-square problem here is formulated as  $f^2(x, \sigma)$  instead of  $\sigma f^2(x)$ .  $\sigma$  denotes the weight.

The results are stored internally to `BaseSolverLeastSquares::_values`.

See Also

`adaptWeights()`, `getValueDimension()`

Definition at line 31 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `teb::BaseSolverLeastSquares::_val_dim`, `teb::BaseSolverLeastSquares::_values`, `teb::BaseSolverLeastSquares::_weight_equalities`, `teb::BaseSolverLeastSquares::_weight_inequalities`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::lsq`, `teb::Config::optim`, `teb::Config::Optim::Solver::Lsq::soft_constr_epsilon`, and `teb::Config::Optim::solver`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

#### 16.33.4.8 void teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ ( ) [protected]

The results are stored to the class member variable `SolverLevenbergMarquardtEigenSparse::_nnz_per_col`.

Remarks

: Make sure that `BaseSolverLeastSquares::_opt_vec_dim` is valid!

Definition at line 311 of file `solver_levenbergmarquardt_eigen_sparse.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, `_nnz_per_col`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, and `teb::VertexType::isFixedComp()`.

Referenced by `allocateSparseJacobian()`.

16.33.4.9 `void teb::BaseSolver::discardBackupVertices ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

16.33.4.10 `double teb::BaseSolverLeastSquares::getChi2 ( ) const` `[inline]`, `[protected]`, `[inherited]`

The  $\chi^2$  error is the scalar cost value of the least-squares optimization problem since the total cost function is composed of weighted terms:

$$f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \mathbf{f}(\mathcal{B})$$

Weights are included in  $\mathbf{f}(\mathcal{B})$ . In this case it is  $\chi^2 = f(\mathcal{B})$ .

**Todo** Maybe switch to the formulation  $f(\mathcal{B}) = \mathbf{f}(\mathcal{B})^T \mathbf{\Omega} \mathbf{f}(\mathcal{B})$  similar to g2o and Teb-Matlab.

Definition at line 82 of file `base_solver_least_squares.h`.

References `teb::BaseSolverLeastSquares::_values`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

16.33.4.11 `int teb::BaseSolver::getDimEqualities ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file `base_solver.h`.

References `teb::BaseSolver::_equalities`.

Referenced by `teb::BaseSolver::solve()`.

16.33.4.12 `int teb::BaseSolver::getDimInequalities ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file `base_solver.h`.

References `teb::BaseSolver::_inequalities`.

Referenced by `teb::BaseSolver::solve()`.

16.33.4.13 `int teb::BaseSolver::getDimObjectives ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file `base_solver.h`.

References `teb::BaseSolver::_objectives`.

Referenced by `teb::BaseSolver::solve()`.

16.33.4.14 `Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const` `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls [VertexType::getDataFree\(\)](#) for each vertex.

## Remarks

Make sure `BaseSolver::_opt_vec_dim` is valid (see `solve()`).

## Returns

`Eigen::Vector` containing all values. The length equals the dimension of all free variables (`getOptVecDimension()`).

Definition at line 162 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_opt_vec_dim`.

**16.33.4.15** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

## See Also

`TebController::getActiveVertices()`, `getValueDimension()`

Definition at line 180 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

**16.33.4.16** `int teb::BaseSolverLeastSquares::getValueDimension ( ) const` `[protected]`, `[inherited]`

This method returns the dimension of the full cost/value vector including all objective and constraints.

In particular it sums up all dimensions of single edges stored in the given hyper-graph.

## Remarks

Make sure `BaseSolver::_objective_dim`, `BaseSolver::_equalities_dim` and `BaseSolver::_inequalities_dim` are valid (see `solve()`).

## Returns

dimension of the complete value/cost vector.

## See Also

`buildValueVector()`, `getOptVecDimension()`, `getDimObjectives()`, `getDimEqualities()`, `getDimInequalities()`

Definition at line 73 of file `base_solver_least_squares.cpp`.

References `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_inequalities_dim`, and `teb::BaseSolver::_objective_dim`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

**16.33.4.17** `virtual void teb::BaseSolverLeastSquares::initWorkspaces ( )` `[inline]`, `[protected]`, `[virtual]`, `[inherited]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each `EdgeType`) or map to existing memory.

Reimplemented from `teb::BaseSolver`.

Definition at line 43 of file `base_solver_least_squares.h`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `PRINT_DEBUG_COND_ONCE`, and `teb::QUADRATIC`.

16.33.4.18 `void teb::BaseSolver::restoreVertices ( ) [inline],[protected],[inherited]`

See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

16.33.4.19 `void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline],[protected],[inherited]`

See Also

[VertexType::top\(\)](#)

Definition at line 227 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

16.33.4.20 `void teb::BaseSolver::setConfig ( const Config * config ) [inline],[inherited]`

Remarks

This function is called from the TEB class within `setSolver` method.

Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file `base_solver.h`.

References `teb::BaseSolver::cfg`.

Referenced by `teb::TebController< p, q >::setSolver()`.

16.33.4.21 `virtual bool teb::BaseSolver::solve ( HyperGraph * optimizable_graph ) [inline],[virtual],[inherited]`

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with [solveImpl\(\)](#).

**Todo** Split `initWorkspaces` into `init` and `update` phase in order to hot-start from previous initializations.

Parameters

<i>optimizable_ - graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
-----------------------------	---

Definition at line 61 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolver::_no_vert_backups`, `teb::BaseSolver::_objective_dim`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::HyperGraph::activeVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::HyperGraph::equalities()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolver::getOptVecDimension()`, `teb::HyperGraph::inequalities()`, `teb::BaseSolver::initWorkspaces()`, `teb::HyperGraph::isGraphModified()`, `teb::HyperGraph::objectives()`, and `teb::BaseSolver::solveImpl()`.

16.33.4.22 `bool teb::SolverLevenbergMarquardtEigenSparse::solveImpl( )` `[protected]`, `[virtual]`

Store results to the vertices of the active vertices container.

Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolverLeastSquares](#).

Definition at line 8 of file `solver_levenbergmarquardt_eigen_sparse.cpp`.

References `_jacobian`, `teb::BaseSolver::_opt_vec_dim`, `_sparse_solver`, `teb::BaseSolverLeastSquares::_val_dim`, `teb::BaseSolverLeastSquares::_values`, `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::backupVertices()`, `buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolverLeastSquares::getChi2()`, `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::restoreVertices()`.

## 16.33.5 Member Data Documentation

16.33.5.1 `VertexContainer* teb::BaseSolver::_active_vertices = nullptr` `[protected]`, `[inherited]`

Definition at line 233 of file `base_solver.h`.

Referenced by `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::applyOptVec()`, `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::getOptVecCopy()`, `teb::BaseSolver::getOptVecDimension()`, `teb::BaseSolver::restoreVertices()`, `teb::BaseSolver::restoreVerticesButKeepBackup()`, and `teb::BaseSolver::solve()`.

16.33.5.2 `EdgeContainer* teb::BaseSolver::_equalities = nullptr` `[protected]`, `[inherited]`

Definition at line 235 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsEq()`, `countJacobianColNNZ()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

16.33.5.3 `int teb::BaseSolver::_equalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 240 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::SolverNloptPackage::getEqConstrDimFromStorage()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

16.33.5.4 `bool teb::BaseSolver::_graph_structure_modified = true` `[protected]`, `[inherited]`

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the [solve\(\)](#) method.

Definition at line 250 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.33.5.5 `EdgeContainer*` `teb::BaseSolver::_inequalities = nullptr` `[protected]`, `[inherited]`

Definition at line 236 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsInEq()`, `countJacobianColNNZ()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

#### 16.33.5.6 `int` `teb::BaseSolver::_inequalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 241 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

#### 16.33.5.7 `Eigen::SparseMatrix<double>` `teb::SolverLevenbergMarquardtEigenSparse::_jacobian` `[protected]`

Definition at line 61 of file `solver_levenbergmarquardt_eigen_sparse.h`.

Referenced by `allocateSparseJacobian()`, `buildJacobian()`, and `solveImpl()`.

#### 16.33.5.8 `Eigen::VectorXi` `teb::SolverLevenbergMarquardtEigenSparse::_nnz_per_col` `[protected]`

Definition at line 63 of file `solver_levenbergmarquardt_eigen_sparse.h`.

Referenced by `allocateSparseJacobian()`, and `countJacobianColNNZ()`.

#### 16.33.5.9 `unsigned int` `teb::BaseSolver::_no_vert_backups = 0` `[protected]`, `[inherited]`

Definition at line 243 of file `base_solver.h`.

Referenced by `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::restoreVertices()`, and `teb::BaseSolver::solve()`.

#### 16.33.5.10 `int` `teb::BaseSolver::_objective_dim = -1` `[protected]`, `[inherited]`

Definition at line 239 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::solve()`.

#### 16.33.5.11 `EdgeContainer*` `teb::BaseSolver::_objectives = nullptr` `[protected]`, `[inherited]`

Definition at line 234 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `countJacobianColNNZ()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::SolverNloptPackage::objectives()`, and `teb::BaseSolver::solve()`.

**16.33.5.12** `int teb::BaseSolver::_opt_vec_dim = -1` `[protected]`, `[inherited]`

Definition at line 238 of file `base_solver.h`.

Referenced by `allocateSparseJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `countJacobianColNNZ()`, `teb::BaseSolver::getOptVecCopy()`, `teb::SolverNloptPackage::getOptVecDimFromStorage()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.33.5.13** `Eigen::SimplicialLDLT<Eigen::SparseMatrix<double>, Eigen::Upper> teb::SolverLevenbergMarquardtEigenSparse::_sparse_solver` `[protected]`

Check [http://eigen.tuxfamily.org/dox/group\\_\\_TopicSparseSystems.html](http://eigen.tuxfamily.org/dox/group__TopicSparseSystems.html) for further information and different solvers. The second template parameter specifies, whether the upper or lower triangular part should be used. If CHOLMOD was found by CMake, the cholmod supernodal llt version is used rather than Eigens simplicial ldt. Define `FORCE_EIGEN_SOLVER` if the default LDLT should still be used even if CHOLMOD is found.

Definition at line 75 of file `solver_levenbergmarquardt_eigen_sparse.h`.

Referenced by `solveImpl()`.

**16.33.5.14** `int teb::BaseSolverLeastSquares::_val_dim = -1` `[protected]`, `[inherited]`

Definition at line 96 of file `base_solver_least_squares.h`.

Referenced by `allocateSparseJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

**16.33.5.15** `Eigen::VectorXd teb::BaseSolverLeastSquares::_values` `[protected]`, `[inherited]`

Definition at line 90 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverLeastSquares::getChi2()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `solveImpl()`.

**16.33.5.16** `int teb::BaseSolverLeastSquares::_weight_adapt_count = 0` `[protected]`, `[inherited]`

Definition at line 92 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`.

**16.33.5.17** `double teb::BaseSolverLeastSquares::_weight_equalities = 1` `[protected]`, `[inherited]`

Definition at line 93 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildJacobian()`, and `teb::BaseSolverLeastSquares::buildValueVector()`.

16.33.5.18 `double teb::BaseSolverLeastSquares::_weight_inequalities = 1` `[protected]`, `[inherited]`

Definition at line 94 of file `base_solver_least_squares.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildJacobian()`, and `teb::BaseSolverLeastSquares::buildValueVector()`.

16.33.5.19 `const Config* teb::BaseSolver::cfg = nullptr` `[inherited]`

See Also

[setConfig\(\)](#)

Definition at line 95 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian-FullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::setConfig()`, and `teb::SolverSQPDense::solveImpl()`.

16.33.5.20 `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW` `[inherited]`

Definition at line 253 of file `base_solver.h`.

The documentation for this class was generated from the following files:

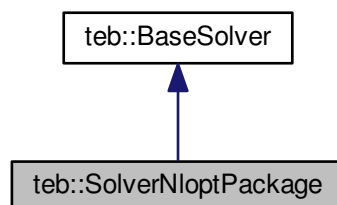
- [solver\\_levenbergmarquardt\\_eigen\\_sparse.h](#)
- [solver\\_levenbergmarquardt\\_eigen\\_sparse.cpp](#)

## 16.34 teb::SolverNloptPackage Class Reference

This class wraps the optimization problem to allow solving by NLOPT.

```
#include <solver_nlopt_package.h>
```

Inheritance diagram for `teb::SolverNloptPackage`:



### Public Types

- using `VertexContainer` = `HyperGraph::VertexContainer`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*



- using [EdgeContainer](#) = [HyperGraph::EdgeContainer](#)  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

## Public Member Functions

- [SolverNloptPackage](#) ()  
*Empty Constructor.*
- virtual void [initWorkspaces](#) ()  
*Initialize workspaces for the block jacobians and hessians.*
- [EdgeContainer](#) \* [objectives](#) ()  
*Return pointer to the edges containing objectives.*
- [EdgeContainer](#) \* [constraintsEq](#) ()  
*Return pointer to the edges containing equalities.*
- [EdgeContainer](#) \* [constraintsInEq](#) ()  
*Return pointer to the edges containing inequalities.*
- int [getOptVecDimFromStorage](#) () const  
*Return dimension of the optimization vector (call [getOptVecDimension\(\)](#) before)*
- int [getEqConstrDimFromStorage](#) () const  
*Return dimension of the equality constraint vector.*
- void [seperateInequalitiesAndBounds](#) ()  
*Seperate constraintsInEq to bound constraints and general nonlinear inequality constraints and store results internally.*
- int [getBoundConstrDimFromStorage](#) () const  
*Return dimension of the equality constraint vector.*
- int [getInEqConstrDimFromStorage](#) () const  
*Return dimension of the inequality constraint vector.*
- void [setConfig](#) (const [Config](#) \*config)  
*Set config including solver settings.*
- virtual bool [solve](#) ([HyperGraph](#) \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

## Static Public Member Functions

- static double [objectiveFunction](#) (unsigned n, const double \*x, double \*grad, void \*solver\_object\_this)  
*Objective function wrapper that iterates all objective edges.*
- static void [equalityConstraintFunction](#) (unsigned m, double \*result, unsigned n, const double \*x, double \*grad, void \*solver\_object\_this)  
*Equality constraint function wrapper that iterates all equality constraint edges.*
- static void [inequalityConstraintFunction](#) (unsigned m, double \*result, unsigned n, const double \*x, double \*grad, void \*solver\_object\_this)  
*Inequality constraint function wrapper that iterates all inequality constraint edges (excluding bound constraints).*

## Public Attributes

- const [Config](#) \* [cfg](#) = nullptr  
*Store pointer to [Config](#) object.*
- [EIGEN\\_MAKE\\_ALIGNED\\_OPERATOR\\_NEW](#)

## Protected Member Functions

- virtual bool [solveImpl](#) ()  
*Solve the actual optimization problem.*
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const  
*Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- [EdgeContainer](#) [\\_bound\\_constraints](#)  
*Store bound constraints using [seperateInequalitiesAndBounds\(\)](#).*
- [EdgeContainer](#) [\\_inequalities\\_without\\_bounds](#)  
*Store general nonlinear constraints using [seperateInequalitiesAndBounds\(\)](#).*
- int [\\_bound\\_constraints\\_dim](#) = 0  
*Store dimension of bound constraints using [seperateInequalitiesAndBounds\(\)](#).*
- int [\\_inequalities\\_without\\_bounds\\_dim](#) = 0  
*Store dimension of general nonlinear constraints using [seperateInequalitiesAndBounds\(\)](#).*
- Eigen::VectorXd [\\_lower\\_bounds](#)  
*Store lower bounds according to the optimization vector.*
- Eigen::VectorXd [\\_upper\\_bounds](#)  
*Store upper bounds according to the optimization vector.*
- Eigen::VectorXd [\\_ctol\\_eq](#)  
*Store tolerances for the equality constraints (stopping criteria)*
- Eigen::VectorXd [\\_ctol\\_ineq](#)  
*Store tolerances for the inequality constraints (stopping criteria)*
- [VertexContainer](#) \* [\\_active\\_vertices](#) = nullptr  
*Pointer to active vertex container (active = non-fixed)*
- [EdgeContainer](#) \* [\\_objectives](#) = nullptr  
*Pointer to edges representing objective functions.*
- [EdgeContainer](#) \* [\\_equalities](#) = nullptr  
*Pointer to edges representing equality constraints.*

- `EdgeContainer * _inequalities = nullptr`  
*Pointer to edges representing inequality constraints.*
- `int _opt_vec_dim = -1`  
*Store dimension of the optimization vector here (see [getOptVecDimension\(\)](#)).*
- `int _objective_dim = -1`  
*Store dimension of the objective function (see [getDimObjectives\(\)](#)).*
- `int _equalities_dim = -1`  
*Store dimension of the equality constraints (see [getDimEqualities\(\)](#)).*
- `int _inequalities_dim = -1`  
*Store dimension of the inequality constraints (see [getDimInequalities\(\)](#)).*
- `unsigned int _no_vert_backups = 0`  
*Track number of vertices backups made.*
- `bool _graph_structure_modified = true`  
*Mark if a new graph structure is available, thus we need to reinitialize all workspaces.*

### 16.34.1 Detailed Description

Wrapper function for the Nlopt optimization library (<http://ab-initio.mit.edu/wiki/index.php/-Nlopt>) The purpose of this solver is not to perform an efficient and fast real-time optimization, rather than having alternative constraint solvers for debugging and testing.

Refer to [Config::Optim::Solver::Nlopt](#) for configuration parameters.

**Test** This class needs more testing (different solvers etc.)

Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

Definition at line 29 of file `solver_nlopt_package.h`.

### 16.34.2 Member Typedef Documentation

16.34.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer` `[inherited]`

Definition at line 36 of file `base_solver.h`.

16.34.2.2 `using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer` `[inherited]`

Definition at line 34 of file `base_solver.h`.

### 16.34.3 Constructor & Destructor Documentation

16.34.3.1 `teb::SolverNloptPackage::SolverNloptPackage ( )` `[inline]`

Definition at line 34 of file `solver_nlopt_package.h`.

### 16.34.4 Member Function Documentation

16.34.4.1 `void teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta )` `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls [VertexType::plusFree\(\)](#) for each vertex.

## Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length equals the dimension of all free variables.
--------------	---

Definition at line 127 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.4.2** `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::setFree\(\)](#) for each vertex.

## Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`.

**16.34.4.3** `void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]`

## See Also

[VertexType::push\(\)](#)

Definition at line 223 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.4.4** `EdgeContainer* teb::SolverNloptPackage::constraintsEq ( ) [inline]`

Definition at line 55 of file solver\_nlopt\_package.h.

References `teb::BaseSolver::_equalities`.

**16.34.4.5** `EdgeContainer* teb::SolverNloptPackage::constraintsInEq ( ) [inline]`

Definition at line 56 of file solver\_nlopt\_package.h.

References `teb::BaseSolver::_inequalities`.

**16.34.4.6** `void teb::BaseSolver::discardBackupVertices ( ) [inline], [protected], [inherited]`

## See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file base\_solver.h.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

16.34.4.7 static void teb::SolverNloptPackage::equalityConstraintFunction ( unsigned *m*, double \* *result*, unsigned *n*, const double \* *x*, double \* *grad*, void \* *solver\_object\_this* ) [static]

16.34.4.8 int teb::SolverNloptPackage::getBoundConstrDimFromStorage ( ) const [inline]

Definition at line 62 of file solver\_nlopt\_package.h.

References `_bound_constraints_dim`.

16.34.4.9 int teb::BaseSolver::getDimEqualities ( ) const [inline],[protected],[inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file base\_solver.h.

References `teb::BaseSolver::_equalities`.

Referenced by `teb::BaseSolver::solve()`.

16.34.4.10 int teb::BaseSolver::getDimInequalities ( ) const [inline],[protected],[inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file base\_solver.h.

References `teb::BaseSolver::_inequalities`.

Referenced by `teb::BaseSolver::solve()`.

16.34.4.11 int teb::BaseSolver::getDimObjectives ( ) const [inline],[protected],[inherited]

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file base\_solver.h.

References `teb::BaseSolver::_objectives`.

Referenced by `teb::BaseSolver::solve()`.

16.34.4.12 int teb::SolverNloptPackage::getEqConstrDimFromStorage ( ) const [inline]

Definition at line 58 of file solver\_nlopt\_package.h.

References `teb::BaseSolver::_equalities_dim`.

16.34.4.13 int teb::SolverNloptPackage::getInEqConstrDimFromStorage ( ) const [inline]

Definition at line 63 of file solver\_nlopt\_package.h.

References `_inequalities_without_bounds_dim`.

16.34.4.14 Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const [inline],[protected],[inherited]

This method iterates the active vertices container and calls [VertexType::getDataFree\(\)](#) for each vertex.

#### Remarks

Make sure [BaseSolver::\\_opt\\_vec\\_dim](#) is valid (see [solve\(\)](#)).

## Returns

Eigen::Vector containing all values. The length equals the dimension of all free variables ([getOptVecDimension\(\)](#)).

Definition at line 162 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_opt_vec_dim`.

**16.34.4.15** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking the previously determined index of the last active vertice stored in the graph.

## See Also

[TebController::getActiveVertices\(\)](#), [getValueDimension\(\)](#)

Definition at line 180 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

**16.34.4.16** `int teb::SolverNloptPackage::getOptVecDimFromStorage ( ) const` `[inline]`

Definition at line 57 of file `solver_nlopt_package.h`.

References `teb::BaseSolver::_opt_vec_dim`.

**16.34.4.17** `static void teb::SolverNloptPackage::inequalityConstraintFunction ( unsigned m, double * result, unsigned n, const double * x, double * grad, void * solver_object_this )` `[static]`

**16.34.4.18** `virtual void teb::SolverNloptPackage::initWorkspaces ( )` `[inline]`, `[virtual]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each [Edge-Type](#)) or map to existing memory.

Reimplemented from [teb::BaseSolver](#).

Definition at line 46 of file `solver_nlopt_package.h`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_inequalities`, and `teb::BaseSolver::_objectives`.

**16.34.4.19** `static double teb::SolverNloptPackage::objectiveFunction ( unsigned n, const double * x, double * grad, void * solver_object_this )` `[static]`

**16.34.4.20** `EdgeContainer* teb::SolverNloptPackage::objectives ( )` `[inline]`

Definition at line 54 of file `solver_nlopt_package.h`.

References `teb::BaseSolver::_objectives`.

**16.34.4.21** `void teb::BaseSolver::restoreVertices ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.34.4.22** `void teb::BaseSolver::restoreVerticesButKeepBackup ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::top\(\)](#)

Definition at line 227 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverSQPDense::solveImpl()`.

**16.34.4.23** `void teb::SolverNloptPackage::seperatelnequalitiesAndBounds ( )`

**16.34.4.24** `void teb::BaseSolver::setConfig ( const Config * config )` `[inline]`, `[inherited]`

Remarks

This function is called from the TEB class within `setSolver` method.

Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file `base_solver.h`.

References `teb::BaseSolver::cfg`.

Referenced by `teb::TebController< p, q >::setSolver()`.

**16.34.4.25** `virtual bool teb::BaseSolver::solve ( HyperGraph * optimizable_graph )` `[inline]`, `[virtual]`, `[inherited]`

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with [solveImpl\(\)](#).

**Todo** Split `initWorkspaces` into `init` and `update` phase in order to hot-start from previous initializations.

Parameters

<i>optimizable_ - graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
-----------------------------	---

Definition at line 61 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolver::_no_vert_backups`, `teb::BaseSolver::_objective_dim`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::HyperGraph::activeVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::HyperGraph::equalities()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolver::getOptVecDimension()`, `teb::HyperGraph::inequalities()`, `teb::BaseSolver::initWorkspaces()`, `teb::HyperGraph::isGraphModified()`, `teb::HyperGraph::objectives()`, and `teb::BaseSolver::solveImpl()`.

16.34.4.26 `virtual bool teb::SolverNloptPackage::solveImpl( )` `[protected]`, `[virtual]`

Store results to the vertices of the active vertices container.

Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolver](#).

## 16.34.5 Member Data Documentation

16.34.5.1 `VertexContainer* teb::BaseSolver::_active_vertices = nullptr` `[protected]`, `[inherited]`

Definition at line 233 of file `base_solver.h`.

Referenced by `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::applyOptVec()`, `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::getOptVecCopy()`, `teb::BaseSolver::getOptVecDimension()`, `teb::BaseSolver::restoreVertices()`, `teb::BaseSolver::restoreVerticesButKeepBackup()`, and `teb::BaseSolver::solve()`.

16.34.5.2 `EdgeContainer teb::SolverNloptPackage::_bound_constraints` `[protected]`

Definition at line 70 of file `solver_nlopt_package.h`.

16.34.5.3 `int teb::SolverNloptPackage::_bound_constraints_dim = 0` `[protected]`

Definition at line 72 of file `solver_nlopt_package.h`.

Referenced by `getBoundConstrDimFromStorage()`.

16.34.5.4 `Eigen::VectorXd teb::SolverNloptPackage::_ctol_eq` `[protected]`

Definition at line 77 of file `solver_nlopt_package.h`.

16.34.5.5 `Eigen::VectorXd teb::SolverNloptPackage::_ctol_ineq` `[protected]`

Definition at line 78 of file `solver_nlopt_package.h`.

16.34.5.6 `EdgeContainer* teb::BaseSolver::_equalities = nullptr` `[protected]`, `[inherited]`

Definition at line 235 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `constraintsEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNZ()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

16.34.5.7 `int teb::BaseSolver::_equalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 240 of file `base_solver.h`.



Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `getEqConstrDimFromStorage()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.5.8** `bool teb::BaseSolver::_graph_structure_modified = true` `[protected]`, `[inherited]`

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the `solve()` method.

Definition at line 250 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.5.9** `EdgeContainer* teb::BaseSolver::_inequalities = nullptr` `[protected]`, `[inherited]`

Definition at line 236 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `constraints::InEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNz()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

**16.34.5.10** `int teb::BaseSolver::_inequalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 241 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverSQPDense::calculateMeritDerivative()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.5.11** `EdgeContainer teb::SolverNloptPackage::_inequalities_without_bounds` `[protected]`

Definition at line 71 of file `solver_nlopt_package.h`.

**16.34.5.12** `int teb::SolverNloptPackage::_inequalities_without_bounds_dim = 0` `[protected]`

Definition at line 73 of file `solver_nlopt_package.h`.

Referenced by `getInEqConstrDimFromStorage()`.

**16.34.5.13** `Eigen::VectorXd teb::SolverNloptPackage::_lower_bounds` `[protected]`

Definition at line 75 of file `solver_nlopt_package.h`.

**16.34.5.14** `unsigned int teb::BaseSolver::_no_vert_backups = 0` `[protected]`, `[inherited]`

Definition at line 243 of file `base_solver.h`.

Referenced by `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::restoreVertices()`, and `teb::BaseSolver::solve()`.

**16.34.5.15** `int teb::BaseSolver::_objective_dim = -1` `[protected]`, `[inherited]`

Definition at line 239 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::solve()`.

**16.34.5.16** `EdgeContainer* teb::BaseSolver::_objectives = nullptr` `[protected]`, `[inherited]`

Definition at line 234 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `initWorkspaces()`, `objectives()`, and `teb::BaseSolver::solve()`.

**16.34.5.17** `int teb::BaseSolver::_opt_vec_dim = -1` `[protected]`, `[inherited]`

Definition at line 238 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getOptVecCopy()`, `getOptVecDimFromStorage()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.5.18** `Eigen::VectorXd teb::SolverNloptPackage::_upper_bounds` `[protected]`

Definition at line 76 of file `solver_nlopt_package.h`.

**16.34.5.19** `const Config* teb::BaseSolver::cfg = nullptr` `[inherited]`

See Also

[setConfig\(\)](#)

Definition at line 95 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `teb::SolverSQPDense::initSolverWorkspace()`, `teb::BaseSolver::setConfig()`, and `teb::SolverSQPDense::solveImpl()`.

**16.34.5.20** `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW` `[inherited]`

Definition at line 253 of file `base_solver.h`.

The documentation for this class was generated from the following file:

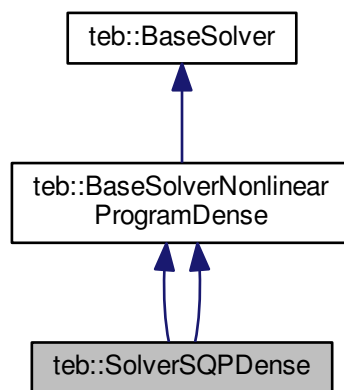
- [solver\\_nlopt\\_package.h](#)

## 16.35 `teb::SolverSQPDense` Class Reference

Dense sequential quadratic programming (SQP) solver for nonlinear programs.

```
#include <solver_sqp_dense.h>
```

Inheritance diagram for `teb::SolverSQPDense`:



### Public Types

- using `VertexContainer` = `HyperGraph::VertexContainer`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using `EdgeContainer` = `HyperGraph::EdgeContainer`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

### Public Member Functions

- `SolverSQPDense` ()  
*Solver Constructor.*
- `SolverSQPDense` ()  
*Empty Constructor.*
- void `setConfig` (const `Config` \*config)  
*Set config including solver settings.*
- virtual bool `solve` (`HyperGraph` \*optimizable\_graph)  
*Solve an optimization problem defined by an hyper-graph.*

## Public Attributes

- const [Config](#) \* [cfg](#) = nullptr  
*Store pointer to [Config](#) object.*
- [EIGEN\\_MAKE\\_ALIGNED\\_OPERATOR\\_NEW](#)

## Protected Types

- using [MatMapRowMajor](#) = Eigen::Map< Eigen::Matrix< double,-1,-1, Eigen::RowMajor >>

## Protected Member Functions

- virtual void [initSolverWorkspace](#) ()
- virtual bool [solveImpl](#) ()  
*Solve the actual optimization problem.*
- bool [checkConvergence](#) ()
- double [calculateMerit](#) (double alpha) const
- double [calculateMeritDerivative](#) (double alpha) const
- virtual void [initSolverWorkspace](#) ()
- virtual bool [solveImpl](#) ()  
*Solve the actual optimization problem.*
- double [calculateMerit](#) (double rho, double \*merit\_derivative\_out=nullptr) const
- virtual void [initWorkspaces](#) ()  
*Initialize workspaces for the block jacobians and hessians.*
- void [initializeLagrangeMultiplier](#) ()
- void [buildObjectiveValue](#) ()  
*Get the sum of all objective values since we solve  $f = \min \sum f_k$ .*
- void [buildEqualityConstraintValueVector](#) ()  
*Build value / cost vector including all equality constraints.*
- void [buildInequalityConstraintValueVector](#) ()  
*Build value / cost vector including all inequality constraints.*
- void [buildObjectiveGradient](#) ()
- void [buildEqualityConstraintJacobian](#) ()
- void [buildInequalityConstraintJacobian](#) ()
- void [calculateLagrangianGradient](#) ()
- void [calculateLagrangianHessian](#) (Eigen::VectorXd \*increment=nullptr)
- void [calculateLagrangianHessianNumerically](#) ()
- void [initHessianBFGS](#) ()
- void [calculateLagrangianHessianFullBFGS](#) (Eigen::VectorXd \*increment)
- void [applyIncrement](#) (const Eigen::Ref< const Eigen::VectorXd > &delta)  
*Apply a new increment obtained by a local optimization to the active vertices.*
- void [applyOptVec](#) (const Eigen::Ref< const Eigen::VectorXd > &opt\_vec)  
*Overwrite TEB states and control inputs with new values.*
- Eigen::VectorXd [getOptVecCopy](#) () const  
*Get a copy of the optimization vector (TEB states, control inputs and dt)*
- int [getOptVecDimension](#) () const  
*Get dimension of the optimization vector (that equals the number of cols and rows in the Hessian).*
- int [getDimObjectives](#) () const  
*Get dimension of the objective function.*
- int [getDimEqualities](#) () const  
*Get dimension of the equality constraints.*
- int [getDimInequalities](#) () const

- *Get dimension of the inequality constraints.*
- void [backupVertices](#) ()  
*Backup all active vertices.*
- void [restoreVertices](#) ()  
*Restore all values of active vertices from the backup stacks.*
- void [restoreVerticesButKeepBackup](#) ()  
*Restore all values of active vertices from the backup stacks WITHOUT discarding the backup.*
- void [discardBackupVertices](#) ()  
*Discard all values of active vertices from the backup stacks.*

## Protected Attributes

- qpOASES::SQProblem [\\_qsolver](#)
- Eigen::Matrix< double,-1,-1,  
Eigen::RowMajor > [\\_A](#)
- Eigen::VectorXd [\\_lb](#)
- Eigen::VectorXd [\\_ub](#)
- Eigen::VectorXd [\\_delta](#)
- Eigen::VectorXd [\\_qdual](#)
- Eigen::Map< Eigen::VectorXd > [\\_dmultiplier\\_eq](#) = Eigen::Map<Eigen::VectorXd>(nullptr,0)
- Eigen::Map< Eigen::VectorXd > [\\_dmultiplier\\_ineq](#) = Eigen::Map<Eigen::VectorXd>(nullptr,0)
- Eigen::VectorXd [\\_merit\\_grad](#)
- Eigen::VectorXd [\\_increment](#)
- double [\\_merit\\_alpha](#) = 0
- double [\\_objective\\_value](#)  
*Store the sum of all values computed in buildObjectiveValueVector() since we solve  $f = \min \sum f_k$ .*
- Eigen::VectorXd [\\_equality\\_values](#)  
*Store the value vector computed in buildEqualityConstraintValueVector().*
- Eigen::VectorXd [\\_inequality\\_values](#)  
*Store the value vector computed in buildInequalityConstraintValueVector().*
- Eigen::VectorXd [\\_objective\\_gradient](#)  
*Store the gradient of the objective value computed in buildObjectiveGradient().*
- MatMapRowMajor [\\_equality\\_jacobian](#) = MatMapRowMajor(nullptr,0,0)  
*Store the jacobian matrix of the equality constraint computed in buildEqualityConstraintJacobian().*
- MatMapRowMajor [\\_inequality\\_jacobian](#) = MatMapRowMajor(nullptr,0,0)  
*Store the jacobian matrix of the inequality constraint computed in buildInequalityConstraintJacobian().*
- Eigen::VectorXd [\\_lagrangian\\_gradient](#)
- Eigen::VectorXd [\\_lagrangian\\_gradient\\_backup](#)  
*The gradient from the last step has to be stored for BFGS.*
- Eigen::Matrix< double,-1,-1,  
Eigen::RowMajor > [\\_lagrangian\\_hessian](#)  
*Store the hessian of the lagrangian  $\nabla^2 L = \nabla^2(f - \mu^T \mathbf{ceq} - \lambda^T \mathbf{c})$ .*
- Eigen::VectorXd [\\_multiplier\\_ineq](#)
- Eigen::VectorXd [\\_multiplier\\_eq](#)
- VertexContainer \* [\\_active\\_vertices](#) = nullptr  
*Pointer to active vertex container (active = non-fixed)*
- EdgeContainer \* [\\_objectives](#) = nullptr  
*Pointer to edges representing objective functions.*
- EdgeContainer \* [\\_equalities](#) = nullptr  
*Pointer to edges representing equality constraints.*
- EdgeContainer \* [\\_inequalities](#) = nullptr  
*Pointer to edges representing inequality constraints.*

- `int _opt_vec_dim = -1`  
Store dimension of the optimization vector here (see [getOptVecDimension\(\)](#)).
- `int _objective_dim = -1`  
Store dimension of the objective function (see [getDimObjectives\(\)](#)).
- `int _equalities_dim = -1`  
Store dimension of the equality constraints (see [getDimEqualities\(\)](#)).
- `int _inequalities_dim = -1`  
Store dimension of the inequality constraints (see [getDimInequalities\(\)](#)).
- `unsigned int _no_vert_backups = 0`  
Track number of vertices backups made.
- `bool _graph_structure_modified = true`  
Mark if a new graph structure is available, thus we need to reinitialize all workspaces.

### 16.35.1 Detailed Description

**Todo** The documentation of this class, after the SQP solver passed a couple of remaining (robustness) tests

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

Definition at line 24 of file solver\_sqp\_dense.h.

### 16.35.2 Member Typedef Documentation

16.35.2.1 `using teb::BaseSolver::EdgeContainer = HyperGraph::EdgeContainer` [inherited]

Definition at line 36 of file base\_solver.h.

16.35.2.2 `using teb::BaseSolverNonlinearProgramDense::MatMapRowMajor = Eigen::Map<Eigen::Matrix<double,-1,-1,Eigen::RowMajor>>` [protected], [inherited]

Definition at line 110 of file base\_solver\_nonlinear\_program\_dense.h.

16.35.2.3 `using teb::BaseSolver::VertexContainer = HyperGraph::VertexContainer` [inherited]

Definition at line 34 of file base\_solver.h.

### 16.35.3 Constructor & Destructor Documentation

16.35.3.1 `teb::SolverSQPDense::SolverSQPDense ( )` [inline]

Definition at line 29 of file solver\_sqp\_dense.h.

References `_qsolver`.

16.35.3.2 `teb::SolverSQPDense::SolverSQPDense ( )` [inline]

Definition at line 19 of file solver\_sqp\_dense\_backup.h.

### 16.35.4 Member Function Documentation

16.35.4.1 `void teb::BaseSolver::applyIncrement ( const Eigen::Ref< const Eigen::VectorXd > & delta )` `[inline]`,  
`[protected]`, `[inherited]`

This method iterates the active vertices container and calls [VertexType::plusFree\(\)](#) for each vertex.

## Parameters

<i>delta</i>	Eigen::Vector containing all increments. The length equals the dimension of all free variables.
--------------	---

Definition at line 127 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `solveImpl()`.

**16.35.4.2** `void teb::BaseSolver::applyOptVec ( const Eigen::Ref< const Eigen::VectorXd > & opt_vec ) [inline], [protected], [inherited]`

This method iterates the active vertices container and calls [VertexType::setFree\(\)](#) for each vertex.

## Parameters

<i>opt_vec</i>	Eigen::Vector containing all values. The length equals the dimension of all free variables.
----------------	---

Definition at line 144 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

**16.35.4.3** `void teb::BaseSolver::backupVertices ( ) [inline], [protected], [inherited]`

## See Also

[VertexType::push\(\)](#)

Definition at line 223 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `solveImpl()`.

**16.35.4.4** `void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian ( ) [protected], [inherited]`

Definition at line 154 of file `base_solver_nonlinear_program_dense.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolverNonlinearProgramDense::_equality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_equality_values`, `teb::BaseSolver::_opt_vec_dim`, `teb::VertexType::dimension()`, `teb::VertexType::getOptVecIdx()`, `teb::VertexType::isFixedAny()`, and `teb::VertexType::isFixedComp()`.

Referenced by `solveImpl()`.

**16.35.4.5** `void teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector ( ) [protected], [inherited]`

Definition at line 30 of file `base_solver_nonlinear_program_dense.cpp`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, and `teb::BaseSolverNonlinearProgramDense::_equality_values`.

Referenced by `solveImpl()`.



**16.35.4.6** void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian ( ) [protected],  
[inherited]

Definition at line 200 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BaseSolver::\_inequalities, teb::BaseSolver::\_inequalities\_dim, teb::BaseSolverNonlinearProgramDense::\_inequality\_jacobian, teb::BaseSolverNonlinearProgramDense::\_inequality\_values, teb::BaseSolver::\_opt\_vec\_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), and teb::VertexType::isFixedComp().

Referenced by solveImpl().

**16.35.4.7** void teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector ( ) [protected],  
[inherited]

Definition at line 54 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BaseSolver::\_inequalities, teb::BaseSolver::\_inequalities\_dim, and teb::BaseSolverNonlinearProgramDense::\_inequality\_values.

Referenced by solveImpl().

**16.35.4.8** void teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient ( ) [protected], [inherited]

Definition at line 82 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::\_objective\_gradient, teb::BaseSolver::\_objectives, teb::BaseSolver::\_opt\_vec\_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb::LINEAR\_SQUARED, and teb::NONLINEAR\_SQUARED.

Referenced by solveImpl().

**16.35.4.9** void teb::BaseSolverNonlinearProgramDense::buildObjectiveValue ( ) [protected], [inherited]

Definition at line 7 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::\_objective\_value, teb::BaseSolver::\_objectives, teb::LINEAR\_SQUARED, and teb::NONLINEAR\_SQUARED.

Referenced by solveImpl().

**16.35.4.10** void teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient ( ) [inline], [protected],  
[inherited]

Definition at line 98 of file base\_solver\_nonlinear\_program\_dense.h.

References teb::BaseSolverNonlinearProgramDense::\_equality\_jacobian, teb::BaseSolverNonlinearProgramDense::\_inequality\_jacobian, teb::BaseSolverNonlinearProgramDense::\_lagrangian\_gradient, teb::BaseSolverNonlinearProgramDense::\_multiplier\_eq, teb::BaseSolverNonlinearProgramDense::\_multiplier\_ineq, and teb::BaseSolverNonlinearProgramDense::\_objective\_gradient.

Referenced by solveImpl().

**16.35.4.11** void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian ( Eigen::VectorXd \* increment =  
nullptr ) [protected], [inherited]

Definition at line 245 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BLOCK\_BFGS, teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS(), teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically(), teb::BaseSolver::cfg,

teb::FULL\_BFGS, teb::FULL\_BFGS\_WITH\_STRUCTURE\_FILTER, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian\_method, teb::Config::Optim::Solver::nonlin\_prog, teb::NUMERIC, teb::Config::optim, PRINT\_ERROR, PRINT\_INFO, teb::Config::Optim::solver, and teb::ZERO\_HESSIAN.

Referenced by solveImpl().

**16.35.4.12** `void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS ( Eigen::VectorXd * increment ) [protected], [inherited]`

Definition at line 520 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BaseSolverNonlinearProgramDense::\_lagrangian\_gradient, teb::BaseSolverNonlinearProgramDense::\_lagrangian\_gradient\_backup, teb::BaseSolverNonlinearProgramDense::\_lagrangian\_hessian, teb::Config::Optim::Solver::NonlinearProgram::Hessian::bfgs\_damped\_mode, teb::BaseSolver::cfg, teb::Config::Optim::Solver::NonlinearProgram::hessian, teb::Config::Optim::Solver::nonlin\_prog, teb::Config::optim, PRINT\_DEBUG, and teb::Config::Optim::solver.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian().

**16.35.4.13** `void teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically ( ) [protected], [inherited]`

Definition at line 273 of file base\_solver\_nonlinear\_program\_dense.cpp.

References teb::BaseSolver::\_equalities, teb::BaseSolver::\_equalities\_dim, teb::BaseSolverNonlinearProgramDense::\_equality\_values, teb::BaseSolver::\_inequalities, teb::BaseSolver::\_inequalities\_dim, teb::BaseSolverNonlinearProgramDense::\_inequality\_values, teb::BaseSolverNonlinearProgramDense::\_lagrangian\_hessian, teb::BaseSolverNonlinearProgramDense::\_multiplier\_eq, teb::BaseSolverNonlinearProgramDense::\_multiplier\_ineq, teb::BaseSolver::\_objectives, teb::BaseSolver::\_opt\_vec\_dim, teb::VertexType::dimension(), teb::VertexType::getOptVecIdx(), teb::VertexType::isFixedAny(), teb::VertexType::isFixedComp(), teb::LINEAR, teb::LINEAR\_SQUARED, teb::NONLINEAR\_ONCE\_DIFF, and teb::NONLINEAR\_SQUARED.

Referenced by teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian(), and teb::BaseSolverNonlinearProgramDense::initHessianBFGS().

**16.35.4.14** `double teb::SolverSQPDense::calculateMerit ( double rho, double * merit_derivative_out = nullptr ) const [inline], [protected]`

Definition at line 205 of file solver\_sqp\_dense\_backup.h.

References \_delta, teb::BaseSolverNonlinearProgramDense::\_equality\_values, teb::BaseSolverNonlinearProgramDense::\_inequality\_values, teb::BaseSolverNonlinearProgramDense::\_objective\_gradient, and teb::BaseSolverNonlinearProgramDense::\_objective\_value.

**16.35.4.15** `double teb::SolverSQPDense::calculateMerit ( double alpha ) const [inline], [protected]`

Definition at line 234 of file solver\_sqp\_dense.h.

References teb::BaseSolverNonlinearProgramDense::\_equality\_values, teb::BaseSolverNonlinearProgramDense::\_inequality\_values, and teb::BaseSolverNonlinearProgramDense::\_objective\_value.

Referenced by solveImpl().

**16.35.4.16** `double teb::SolverSQPDense::calculateMeritDerivative ( double alpha ) const [inline], [protected]`

Definition at line 243 of file solver\_sqp\_dense.h.

References `_delta`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolverNonlinearProgramDense::_equality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_equality_values`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolverNonlinearProgramDense::_inequality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_inequality_values`, and `teb::BaseSolverNonlinearProgramDense::_objective_gradient`.

Referenced by `solveImpl()`.

**16.35.4.17** `bool teb::SolverSQPDense::checkConvergence ( )` `[inline]`, `[protected]`

Definition at line 228 of file `solver_sqp_dense.h`.

References `teb::BaseSolverNonlinearProgramDense::_equality_values`, and `teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient`.

Referenced by `solveImpl()`.

**16.35.4.18** `void teb::BaseSolver::discardBackupVertices ( )` `[inline]`, `[protected]`, `[inherited]`

See Also

[VertexType::discardTop\(\)](#)

Definition at line 229 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `solveImpl()`.

**16.35.4.19** `int teb::BaseSolver::getDimEqualities ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 202 of file `base_solver.h`.

References `teb::BaseSolver::_equalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.35.4.20** `int teb::BaseSolver::getDimInequalities ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 214 of file `base_solver.h`.

References `teb::BaseSolver::_inequalities`.

Referenced by `teb::BaseSolver::solve()`.

**16.35.4.21** `int teb::BaseSolver::getDimObjectives ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking each edge dimension.

Definition at line 190 of file `base_solver.h`.

References `teb::BaseSolver::_objectives`.

Referenced by `teb::BaseSolver::solve()`.

**16.35.4.22** `Eigen::VectorXd teb::BaseSolver::getOptVecCopy ( ) const` `[inline]`, `[protected]`, `[inherited]`

This method iterates the active vertices container and calls [VertexType::getDataFree\(\)](#) for each vertex.

## Remarks

Make sure `BaseSolver::_opt_vec_dim` is valid (see `solve()`).

## Returns

`Eigen::Vector` containing all values. The length equals the dimension of all free variables (`getOptVecDimension()`).

Definition at line 162 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_opt_vec_dim`.

**16.35.4.23** `int teb::BaseSolver::getOptVecDimension ( ) const` `[inline]`, `[protected]`, `[inherited]`

The dimension is obtained by checking the previously determined index of the last active vertex stored in the graph.

## See Also

`TebController::getActiveVertices()`, `getValueDimension()`

Definition at line 180 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `teb::BaseSolver::solve()`.

**16.35.4.24** `void teb::BaseSolverNonlinearProgramDense::initHessianBFGS ( )` `[protected]`, `[inherited]`

Definition at line 489 of file `base_solver_nonlinear_program_dense.cpp`.

References `teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient`, `teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup`, `teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian`, `teb::BaseSolver::_opt_vec_dim`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::NonlinearProgram::hessian`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_init_identity_scale`, `teb::IDENTITY`, `teb::Config::Optim::Solver::nonlin_prog`, `teb::NUMERIC`, `teb::Config::optim`, `PRINT_ERROR`, `teb::Config::Optim::solver`, and `teb::ZERO`.

Referenced by `solveImpl()`.

**16.35.4.25** `void teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier ( )` `[inline]`, `[protected]`, `[inherited]`

Definition at line 70 of file `base_solver_nonlinear_program_dense.h`.

References `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolverNonlinearProgramDense::_multiplier_eq`, and `teb::BaseSolverNonlinearProgramDense::_multiplier_ineq`.

Referenced by `solveImpl()`.

**16.35.4.26** `virtual void teb::SolverSQPDense::initSolverWorkspace ( )` `[inline]`, `[protected]`, `[virtual]`

Implements `teb::BaseSolverNonlinearProgramDense`.

Definition at line 24 of file `solver_sqp_dense_backup.h`.

References `_A`, `_delta`, `_dmultiplier_eq`, `_dmultiplier_ineq`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolverNonlinearProgramDense::_equality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_equality_values`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolverNonlinearProgramDense::_inequality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_inequality_values`, `_lb`, `_merit_grad`, `teb::BaseSolver::_opt_vec_dim`, `_qdual`, `_ub`, and `INF`.

**16.35.4.27** `virtual void teb::SolverSQPDense::initSolverWorkspace ( ) [inline],[protected],[virtual]`

Implements [teb::BaseSolverNonlinearProgramDense](#).

Definition at line 37 of file `solver_sqp_dense.h`.

References `_A`, `_delta`, `_dmultiplier_eq`, `_dmultiplier_ineq`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolverNonlinearProgramDense::_equality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_equality_values`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolverNonlinearProgramDense::_inequality_jacobian`, `teb::BaseSolverNonlinearProgramDense::_inequality_values`, `_lb`, `_merit_alpha`, `_merit_grad`, `teb::BaseSolver::_opt_vec_dim`, `_qdual`, `_qsolver`, `_ub`, `teb::Config::Optim::Solver::NonlinearProgram::LineSearch::alpha_init`, `teb::BaseSolver::cfg`, `teb::Config::Optim::Solver::NonlinearProgram::hessian`, `teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian_method`, `INF`, `teb::Config::Optim::Solver::NonlinearProgram::linesearch`, `teb::Config::Optim::Solver::nonlin_prog`, `teb::Config::optim`, `teb::Config::Optim::solver`, and `teb::ZERO_HESSIAN`.

Referenced by `solveImpl()`.

**16.35.4.28** `virtual void teb::BaseSolverNonlinearProgramDense::initWorkspaces ( ) [inline],[protected],[virtual],[inherited]`

Implement this function to allocate new memory for the jacobian and hessian workspaces (as part of each [EdgeType](#)) or map to existing memory.

Reimplemented from [teb::BaseSolver](#).

Definition at line 46 of file `base_solver_nonlinear_program_dense.h`.

References `teb::BaseSolver::_equalities`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_objectives`, `teb::LINEAR_SQUARED`, `teb::NONLINEAR_SQUARED`, and `PRINT_DEBUG_COND_ONCE`.

**16.35.4.29** `void teb::BaseSolver::restoreVertices ( ) [inline],[protected],[inherited]`

See Also

[VertexType::pop\(\)](#)

Definition at line 225 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, and `teb::BaseSolver::_no_vert_backups`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, and `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`.

**16.35.4.30** `void teb::BaseSolver::restoreVerticesButKeepBackup ( ) [inline],[protected],[inherited]`

See Also

[VertexType::top\(\)](#)

Definition at line 227 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`.

Referenced by `solveImpl()`.

**16.35.4.31** `void teb::BaseSolver::setConfig ( const Config * config ) [inline],[inherited]`

Remarks

This function is called from the TEB class within `setSolver` method.

## Parameters

<i>config</i>	Pointer to <a href="#">Config</a> object.
---------------	---

Definition at line 49 of file `base_solver.h`.

References `teb::BaseSolver::cfg`.

Referenced by `teb::TebController< p, q >::setSolver()`.

**16.35.4.32** `virtual bool teb::BaseSolver::solve ( HyperGraph * optimizable_graph )` `[inline]`, `[virtual]`, `[inherited]`

The method copies pointers to the active vertices and edges (objectives, equality constraints and inequality constraints) and calls the actual solver with `solveImpl()`.

**Todo** Split `initWorkspaces` into `init` and `update` phase in order to hot-start from previous initializations.

## Parameters

<i>optimizable_graph</i>	pointer to the <a href="#">HyperGraph</a> that should be solved
--------------------------	---

Definition at line 61 of file `base_solver.h`.

References `teb::BaseSolver::_active_vertices`, `teb::BaseSolver::_equalities`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolver::_graph_structure_modified`, `teb::BaseSolver::_inequalities`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolver::_no_vert_backups`, `teb::BaseSolver::_objective_dim`, `teb::BaseSolver::_objectives`, `teb::BaseSolver::_opt_vec_dim`, `teb::HyperGraph::activeVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::HyperGraph::equalities()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolver::getOptVecDimension()`, `teb::HyperGraph::inequalities()`, `teb::BaseSolver::initWorkspaces()`, `teb::HyperGraph::isGraphModified()`, `teb::HyperGraph::objectives()`, and `teb::BaseSolver::solveImpl()`.

**16.35.4.33** `virtual bool teb::SolverSQPDense::solveImpl ( )` `[inline]`, `[protected]`, `[virtual]`

Store results to the vertices of the active vertices container.

## Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolverNonlinearProgramDense](#).

Definition at line 63 of file `solver_sqp_dense_backup.h`.

References `_A`, `_delta`, `_dmultiplier_eq`, `_dmultiplier_ineq`, `teb::BaseSolver::_equalities_dim`, `teb::BaseSolverNonlinearProgramDense::_equality_values`, `teb::BaseSolver::_inequalities_dim`, `teb::BaseSolverNonlinearProgramDense::_inequality_values`, `teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian`, `_lb`, `teb::BaseSolverNonlinearProgramDense::_multiplier_eq`, `teb::BaseSolverNonlinearProgramDense::_multiplier_ineq`, `teb::BaseSolverNonlinearProgramDense::_objective_gradient`, `teb::BaseSolver::_opt_vec_dim`, `_q dual`, `_ub`, `teb::BaseSolver::applyIncrement()`, `teb::BaseSolver::backupVertices()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `calculateMerit()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `initSolverWorkspace()`, `PRINT_DEBUG`, and `teb::BaseSolver::restoreVerticesButKeepBackup()`.

16.35.4.34 virtual bool teb::SolverSQPDense::solveImpl ( ) [inline],[protected],[virtual]

Store results to the vertices of the active vertices container.

## Return values

<i>true</i>	optimization was successfull.
<i>false</i>	optimization was not successfull.

Implements [teb::BaseSolverNonlinearProgramDense](#).

Definition at line 86 of file solver\_sqp\_dense.h.

References [\\_A](#), [\\_delta](#), [\\_dmultiplier\\_eq](#), [\\_dmultiplier\\_ineq](#), [teb::BaseSolver::\\_equalities\\_dim](#), [teb::BaseSolverNonlinearProgramDense::\\_equality\\_values](#), [teb::BaseSolver::\\_graph\\_structure\\_modified](#), [\\_increment](#), [teb::BaseSolver::\\_inequalities\\_dim](#), [teb::BaseSolverNonlinearProgramDense::\\_inequality\\_values](#), [teb::BaseSolverNonlinearProgramDense::\\_lagrangian\\_hessian](#), [\\_lb](#), [\\_merit\\_alpha](#), [teb::BaseSolverNonlinearProgramDense::\\_multiplier\\_eq](#), [teb::BaseSolverNonlinearProgramDense::\\_multiplier\\_ineq](#), [teb::BaseSolverNonlinearProgramDense::\\_objective\\_gradient](#), [\\_qdual](#), [\\_qsolver](#), [\\_ub](#), [teb::BaseSolver::applyIncrement\(\)](#), [teb::BaseSolver::backupVertices\(\)](#), [teb::Config::Optim::Solver::NonlinearProgram::LineSearch::beta](#), [teb::BLOCK\\_BFGS](#), [teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient\(\)](#), [teb::BaseSolverNonlinearProgramDense::buildObjectiveValue\(\)](#), [teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient\(\)](#), [teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian\(\)](#), [calculateMerit\(\)](#), [calculateMeritDerivative\(\)](#), [teb::BaseSolver::cfg](#), [checkConvergence\(\)](#), [teb::BaseSolver::discardBackupVertices\(\)](#), [teb::FULL\\_BFGS](#), [teb::FULL\\_BFGS\\_WITH\\_STRUCTURE\\_FILTER](#), [teb::Config::Optim::Solver::NonlinearProgram::hessian](#), [teb::Config::Optim::Solver::NonlinearProgram::Hessian::hessian\\_method](#), [teb::BaseSolverNonlinearProgramDense::initHessianBFGS\(\)](#), [teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier\(\)](#), [initSolverWorkspace\(\)](#), [teb::Config::Optim::Solver::NonlinearProgram::linesearch](#), [teb::Config::Optim::Solver::nonlin\\_prog](#), [teb::Config::optim](#), [PRINT\\_DEBUG](#), [PRINT\\_DEBUG\\_ONCE](#), [teb::BaseSolver::restoreVerticesButKeepBackup\(\)](#), [teb::Config::Optim::Solver::NonlinearProgram::LineSearch::sigma](#), and [teb::Config::Optim::solver](#).

## 16.35.5 Member Data Documentation

**16.35.5.1** `Eigen::Matrix< double,-1,-1, Eigen::RowMajor > teb::SolverSQPDense::_A` `[protected]`

Definition at line 284 of file solver\_sqp\_dense.h.

Referenced by [initSolverWorkspace\(\)](#), and [solveImpl\(\)](#).

**16.35.5.2** `VertexContainer* teb::BaseSolver::_active_vertices = nullptr` `[protected]`, `[inherited]`

Definition at line 233 of file base\_solver.h.

Referenced by [teb::BaseSolver::applyIncrement\(\)](#), [teb::BaseSolver::applyOptVec\(\)](#), [teb::BaseSolver::backupVertices\(\)](#), [teb::BaseSolver::discardBackupVertices\(\)](#), [teb::BaseSolver::getOptVecCopy\(\)](#), [teb::BaseSolver::getOptVecDimension\(\)](#), [teb::BaseSolver::restoreVertices\(\)](#), [teb::BaseSolver::restoreVerticesButKeepBackup\(\)](#), and [teb::BaseSolver::solve\(\)](#).

**16.35.5.3** `Eigen::VectorXd teb::SolverSQPDense::_delta` `[protected]`

Definition at line 288 of file solver\_sqp\_dense.h.

Referenced by [calculateMerit\(\)](#), [calculateMeritDerivative\(\)](#), [initSolverWorkspace\(\)](#), and [solveImpl\(\)](#).

**16.35.5.4** `Eigen::Map< Eigen::VectorXd > teb::SolverSQPDense::_dmultiplier_eq = Eigen::Map<Eigen::VectorXd>(nullptr,0)` `[protected]`

Definition at line 290 of file solver\_sqp\_dense.h.

Referenced by [initSolverWorkspace\(\)](#), and [solveImpl\(\)](#).



**16.35.5.5** `Eigen::Map< Eigen::VectorXd > teb::SolverSQPDense::_dmultiplier_ineq = Eigen::Map<Eigen::VectorXd>(nullptr,0)`  
`[protected]`

Definition at line 291 of file solver\_sqp\_dense.h.

Referenced by `initSolverWorkspace()`, and `solveImpl()`.

**16.35.5.6** `EdgeContainer* teb::BaseSolver::_equalities = nullptr` `[protected]`, `[inherited]`

Definition at line 235 of file base\_solver.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`, `teb::BaseSolver::getDimEqualities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

**16.35.5.7** `int teb::BaseSolver::_equalities_dim = -1` `[protected]`, `[inherited]`

Definition at line 240 of file base\_solver.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `calculateMeritDerivative()`, `teb::SolverNloptPackage::getEqConstrDimFromStorage()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `solveImpl()`.

**16.35.5.8** `MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_equality_jacobian = MatMapRowMajor(nullptr,0,0)`  
`[protected]`, `[inherited]`

Definition at line 117 of file base\_solver\_nonlinear\_program\_dense.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient()`, `calculateMeritDerivative()`, and `initSolverWorkspace()`.

**16.35.5.9** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_equality_values` `[protected]`, `[inherited]`

Definition at line 113 of file base\_solver\_nonlinear\_program\_dense.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `calculateMerit()`, `calculateMeritDerivative()`, `checkConvergence()`, `initSolverWorkspace()`, and `solveImpl()`.

**16.35.5.10** `bool teb::BaseSolver::_graph_structure_modified = true` `[protected]`, `[inherited]`

In addition some solver can implement hotstarting and decide whether hotstart or not using this flag. It is obtained from the graph passed to the `solve()` method.

Definition at line 250 of file base\_solver.h.

Referenced by `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::BaseSolver::solve()`, and `solveImpl()`.

**16.35.5.11** `Eigen::VectorXd teb::SolverSQPDense::_increment` [protected]

Definition at line 294 of file `solver_sqp_dense.h`.

Referenced by `solveImpl()`.

**16.35.5.12** `EdgeContainer* teb::BaseSolver::_inequalities = nullptr` [protected], [inherited]

Definition at line 236 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverNloptPackage::constraintsInEq()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNZ()`, `teb::BaseSolver::getDimInequalities()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, and `teb::BaseSolver::solve()`.

**16.35.5.13** `int teb::BaseSolver::_inequalities_dim = -1` [protected], [inherited]

Definition at line 241 of file `base_solver.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `calculateMeritDerivative()`, `teb::BaseSolverLeastSquares::getValueDimension()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, `initSolverWorkspace()`, `teb::BaseSolver::solve()`, and `solveImpl()`.

**16.35.5.14** `MatMapRowMajor teb::BaseSolverNonlinearProgramDense::_inequality_jacobian = MatMapRowMajor(nullptr,0,0)` [protected], [inherited]

Definition at line 118 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient()`, `calculateMeritDerivative()`, and `initSolverWorkspace()`.

**16.35.5.15** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_inequality_values` [protected], [inherited]

Definition at line 114 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `calculateMerit()`, `calculateMeritDerivative()`, `initSolverWorkspace()`, and `solveImpl()`.

**16.35.5.16** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient` [protected], [inherited]

Definition at line 120 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `checkConvergence()`, and `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`.

**16.35.5.17** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_lagrangian_gradient_backup` `[protected]`, `[inherited]`

Definition at line 121 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, and `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`.

**16.35.5.18** `Eigen::Matrix<double,-1,-1,Eigen::RowMajor> teb::BaseSolverNonlinearProgramDense::_lagrangian_hessian` `[protected]`, `[inherited]`

Definition at line 122 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, and `solveImpl()`.

**16.35.5.19** `Eigen::VectorXd teb::SolverSQPDense::_lb` `[protected]`

Definition at line 285 of file `solver_sqp_dense.h`.

Referenced by `initSolverWorkspace()`, and `solveImpl()`.

**16.35.5.20** `double teb::SolverSQPDense::_merit_alpha = 0` `[protected]`

Definition at line 296 of file `solver_sqp_dense.h`.

Referenced by `initSolverWorkspace()`, and `solveImpl()`.

**16.35.5.21** `Eigen::VectorXd teb::SolverSQPDense::_merit_grad` `[protected]`

Definition at line 292 of file `solver_sqp_dense.h`.

Referenced by `initSolverWorkspace()`.

**16.35.5.22** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_eq` `[protected]`, `[inherited]`

Definition at line 125 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, and `solveImpl()`.

**16.35.5.23** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_multiplier_ineq` `[protected]`, `[inherited]`

Definition at line 124 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::BaseSolverNonlinearProgramDense::initializeLagrangeMultiplier()`, and `solveImpl()`.

**16.35.5.24** `unsigned int teb::BaseSolver::_no_vert_backups = 0` `[protected]`, `[inherited]`

Definition at line 243 of file `base_solver.h`.

Referenced by `teb::BaseSolver::backupVertices()`, `teb::BaseSolver::discardBackupVertices()`, `teb::BaseSolver::restoreVertices()`, and `teb::BaseSolver::solve()`.

**16.35.5.25** `int teb::BaseSolver::_objective_dim = -1` [protected], [inherited]

Definition at line 239 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::getValueDimension()`, and `teb::BaseSolver::solve()`.

**16.35.5.26** `Eigen::VectorXd teb::BaseSolverNonlinearProgramDense::_objective_gradient` [protected], [inherited]

Definition at line 116 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianGradient()`, `calculateMerit()`, `calculateMeritDerivative()`, and `solveImpl()`.

**16.35.5.27** `double teb::BaseSolverNonlinearProgramDense::_objective_value` [protected], [inherited]

Definition at line 112 of file `base_solver_nonlinear_program_dense.h`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, and `calculateMerit()`.

**16.35.5.28** `EdgeContainer* teb::BaseSolver::_objectives = nullptr` [protected], [inherited]

Definition at line 234 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveValue()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNz()`, `teb::BaseSolver::getDimObjectives()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, `teb::SolverNloptPackage::initWorkspaces()`, `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`, `teb::SolverNloptPackage::objectives()`, and `teb::BaseSolver::solve()`.

**16.35.5.29** `int teb::BaseSolver::_opt_vec_dim = -1` [protected], [inherited]

Definition at line 238 of file `base_solver.h`.

Referenced by `teb::SolverLevenbergMarquardtEigenSparse::allocateSparseJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColINNz()`, `teb::BaseSolver::getOptVecCopy()`, `teb::SolverNloptPackage::getOptVecDimFromStorage()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `initSolverWorkspace()`, `teb::BaseSolver::solve()`, `teb::SolverLevenbergMarquardtEigenDense::solveImpl()`, `teb::SolverLevenbergMarquardtEigenSparse::solveImpl()`, and `solveImpl()`.

**16.35.5.30** `Eigen::VectorXd teb::SolverSQPDense::_qdual` [protected]

Definition at line 289 of file `solver_sqp_dense.h`.

Referenced by `initSolverWorkspace()`, and `solveImpl()`.

**16.35.5.31** `qpOASES::SQProblem teb::SolverSQPDense::_qsolver` [protected]

Definition at line 282 of file `solver_sqp_dense.h`.

Referenced by `initSolverWorkspace()`, `solveImpl()`, and `SolverSQPDense()`.

16.35.5.32 `Eigen::VectorXd teb::SolverSQPDense::_ub` `[protected]`

Definition at line 286 of file `solver_sq_dense.h`.

Referenced by `initSolverWorkspace()`, and `solveImpl()`.

16.35.5.33 `const Config* teb::BaseSolver::cfg = nullptr` `[inherited]`

See Also

[setConfig\(\)](#)

Definition at line 95 of file `base_solver.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverLeastSquares::buildValueVector()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`, `initSolverWorkspace()`, `teb::BaseSolver::setConfig()`, and `solveImpl()`.

16.35.5.34 `teb::BaseSolver::EIGEN_MAKE_ALIGNED_OPERATOR_NEW` `[inherited]`

Definition at line 253 of file `base_solver.h`.

The documentation for this class was generated from the following files:

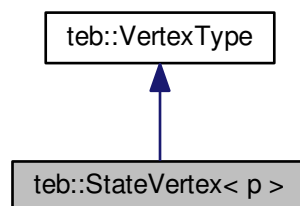
- [solver\\_sq\\_dense.h](#)
- [solver\\_sq\\_dense\\_backup.h](#)

## 16.36 `teb::StateVertex< p >` Class Template Reference

Vertex that contains the `StateVector`.

```
#include <teb_vertices.h>
```

Inheritance diagram for `teb::StateVertex< p >`:



### Public Types

- using [StateVector](#) = `Eigen::Matrix< double, p, 1 >`  
*Typedef for state vector with p states [p x 1].*
- using [ScalarType](#) = `Eigen::Matrix< double, 1, 1 >`

- *Typedef for a 1D-scalar value represented as Eigen::Matrix type [1 x 1].*  
 using `FixedStates` = Eigen::Matrix< bool, p, 1 >  
*Typedef for logical mask that stores fixed and unfixed states [p x 1].*
- using `BackupStackType` = std::stack< `StateVector`, std::vector< `StateVector`, Eigen::aligned\_allocator< `StateVector` > > >  
*Typedef for the backup stack.*
- using `VertexContainer` = std::vector< `VertexType` \* >  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*

## Public Member Functions

- `StateVertex` ()  
*Default constructor: states vector initialized to zero.*
- `StateVertex` (const Eigen::Ref< const `StateVector` > &states)  
*Construct `StateVertex` with predefined state values.*
- `StateVertex` (const double \*states)  
*Construct `StateVertex` with predefined state values and predefined control inputs.*
- `StateVertex` (const `StateVertex`< p > &obj)  
*Copy constructor.*
- virtual int `dimension` () const  
*Return number of elements/values/components stored in this vertex.*
- virtual int `dimensionFree` () const  
*Return only the dimension of free (unfixed) variables.*
- virtual void `plus` (const `VertexType` \*rhs)  
*Define the increment for the vertex:  $x = x + rhs$ .*
- virtual void `plus` (const double \*rhs)  
*Define the increment for the vertex:  $x = x + rhs$  (rhs[] with `dimension()`=p+q values)*
- virtual void `plusFree` (const double \*rhs)  
*Define the increment for the vertex:  $x = x + rhs$  (But only add FREE variables, rhs[] with `dimensionFree()` values).*
- virtual void `setFree` (const double \*rhs)  
*Overwrite all free variables with the values given by rhs (rhs[] with `dimensionFree()` values).*
- virtual void `getDataFree` (double \*target\_vec) const  
*Return a copy of the free (unfixed) values as a single array ( length: `dimensionFree()` ).*
- virtual const double & `getData` (unsigned int idx) const  
*Return pointer to data with the index `idx` ).*
- const `StateVector` & `states` () const  
*Get StateVector (read-only).*
- `StateVector` & `states` ()  
*Get StateVector.*
- virtual void `setStates` (const Eigen::Ref< const `StateVector` > &s)  
*Set or update StateVector.*
- virtual void `setStates` (double state)  
*Set or update only first component of the StateVector (useful for 1d states).*
- const `FixedStates` & `fixed_states` () const  
*Get logical map that defines fixed and unfixed states (fixed = true).*
- void `setFixedAll` (bool fixed)  
*Set all states of this object fixed or unfixed.*
- void `setFixedState` (unsigned int state\_idx, bool fixed)  
*Set a single component of the StateVector to fixed/unfixed.*
- void `setFixedStates` (bool fixed)

- *Set all states of the StateVector to fixed/unfixed.*
- void [setFixedStates](#) (const bool \*fixed)
  - *Set selected states of the StateVector to fixed/unfixed at once.*
- void [setFixedStates](#) (const Eigen::Ref< const [FixedStates](#) > &fixed\_states)
  - *Set selected states of the StateVector to fixed/unfixed at once (Eigen version).*
- virtual bool [isFixedAll](#) () const
  - *return true, if ALL values of this vertex are fixed and therefore are not necessary for optimization.*
- bool [isFixedAny](#) () const
  - *return true, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.*
- virtual bool [isFixedComp](#) (int idx) const
  - *Check if selected element of the StateVector is fixed.*
- virtual void [push](#) ()
  - *This function should store all values into a internal backup stack.*
- virtual void [pop](#) ()
  - *This function should restore the previously stored values of the backup stack and remove them from the stack.*
- virtual void [top](#) ()
  - *This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.*
- virtual void [discardTop](#) ()
  - *This function should delete the previously made backup from the stack without restoring it.*
- virtual unsigned int [stackSize](#) () const
  - *This function should return the current size/number of backups of the backup stack.*
- [StateVertex](#)< p > & [operator=](#) (const [StateVertex](#)< p > &rhs)
- void [setOptVecIdx](#) (int opt\_vec\_idx)
  - *Update the position of the first value of this vertex inside the optimization vector / Hessian.*
- int [getOptVecIdx](#) () const
  - *Get the position of the first value of this Vertex inside the optimization vector / Hessian.*

## Static Public Attributes

- static const int [DimStates](#) = p
  - *Number of states / Size of the state vector as static variable.*
- static const int [Dimension](#) = p
  - *Number of total values stored in this vertex (number of state vector components).*

## Protected Attributes

- [StateVector](#) \_states
  - *State vector with p states [p x 1].*
- [BackupStackType](#) \_backup
  - *backup stack (store and restore states).*
- int [\\_dimension](#) = p
  - *Store dimension for all values stored (p states)*
- [FixedStates](#) \_fixed\_states = FixedStates::Constant(false)
  - *Mask that stores fixed and unfixed states [p x 1] true: fixed, false: free.*
- int [\\_opt\\_vec\\_idx](#) = -1
  - *Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))*

## Friends

- bool `operator==` (const [StateVertex](#)< p > &lhs, const [StateVertex](#)< p > &rhs)
- bool `operator!=` (const [StateVertex](#)< p > &lhs, const [StateVertex](#)< p > &rhs)
- `std::ostream & operator<<` (`std::ostream &os`, const [StateVertex](#)< p > &state)

*Print state vector using `std::ostream`.*

### 16.36.1 Detailed Description

```
template<int p>class teb::StateVertex< p >
```

This class wraps the discrete state vector  $\mathbf{x}_k \in \mathbb{R}^p$  into a vertex type for the hyper-graph.

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

[ControlVertex](#), [VertexType](#), [TimeDiff](#), [BaseEdge](#)

#### Template Parameters

<i>p</i>	Number of states / Size of the state vector
----------	---

Definition at line 25 of file `teb_vertices.h`.

### 16.36.2 Member Typedef Documentation

**16.36.2.1** `template<int p> using teb::StateVertex< p >::BackupStackType = std::stack<StateVector, std::vector<StateVector, Eigen::aligned_allocator<StateVector> > >`

Definition at line 42 of file `teb_vertices.h`.

**16.36.2.2** `template<int p> using teb::StateVertex< p >::FixedStates = Eigen::Matrix<bool, p, 1>`

Definition at line 36 of file `teb_vertices.h`.

**16.36.2.3** `template<int p> using teb::StateVertex< p >::ScalarType = Eigen::Matrix<double, 1, 1>`

Definition at line 34 of file `teb_vertices.h`.

**16.36.2.4** `template<int p> using teb::StateVertex< p >::StateVector = Eigen::Matrix<double, p, 1>`

Definition at line 32 of file `teb_vertices.h`.

**16.36.2.5** `using teb::VertexType::VertexContainer = std::vector<VertexType*> [inherited]`

Definition at line 33 of file `graph.h`.



### 16.36.3 Constructor & Destructor Documentation

16.36.3.1 `template<int p> teb::StateVertex< p >::StateVertex ( ) [inline]`

Definition at line 50 of file `teb_vertices.h`.

16.36.3.2 `template<int p> teb::StateVertex< p >::StateVertex ( const Eigen::Ref< const StateVector > & states ) [inline]`

Parameters

<i>states</i>	Eigen::Vector with p states.
---------------	------------------------------

Definition at line 56 of file `teb_vertices.h`.

16.36.3.3 `template<int p> teb::StateVertex< p >::StateVertex ( const double * states ) [inline]`

Parameters

<i>states</i>	double array with p states.
---------------	-----------------------------

Definition at line 62 of file `teb_vertices.h`.

16.36.3.4 `template<int p> teb::StateVertex< p >::StateVertex ( const StateVertex< p > & obj ) [inline]`

Parameters

<i>obj</i>	Object of type <a href="#">StateVertex</a> *
------------	--

Definition at line 70 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`, and `teb::StateVertex< p >::fixed_states()`.

### 16.36.4 Member Function Documentation

16.36.4.1 `template<int p> virtual int teb::StateVertex< p >::dimension ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 79 of file `teb_vertices.h`.

16.36.4.2 `template<int p> virtual int teb::StateVertex< p >::dimensionFree ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 81 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`.

16.36.4.3 `template<int p> virtual void teb::StateVertex< p >::discardTop ( ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 213 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_backup`.

**16.36.4.4** `template<int p> const FixedStates& teb::StateVertex< p >::fixed_states ( ) const [inline]`

#### Returns

logical map (p x 1 vector of booleans)

Definition at line 152 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`.

Referenced by `teb::StateVertex< p >::operator=()`, `teb::StateVertex< p >::setFixedStates()`, and `teb::StateVertex< p >::StateVertex()`.

**16.36.4.5** `template<int p> virtual const double& teb::StateVertex< p >::getData ( unsigned int idx ) const [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 139 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`.

**16.36.4.6** `template<int p> virtual void teb::StateVertex< p >::getDataFree ( double * target_vec ) const [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 126 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`, and `teb::StateVertex< p >::isFixedComp()`.

**16.36.4.7** `int teb::VertexType::getOptVecIdx ( ) const [inline], [inherited]`

#### Returns

position of the first element of this vertex.

Definition at line 59 of file `graph.h`.

References `teb::VertexType::_opt_vec_idx`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, and `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`.

**16.36.4.8** `template<int p> virtual bool teb::StateVertex< p >::isFixedAll ( ) const [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 190 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`.

**16.36.4.9** `template<int p> bool teb::StateVertex< p >::isFixedAny ( ) const [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 191 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`.

**16.36.4.10** `template<int p> virtual bool teb::StateVertex< p >::isFixedComp ( int idx ) const` `[inline],`  
`[virtual]`

#### Parameters

<i>idx</i>	index to element in the StateVector
------------	-------------------------------------

#### Returns

`true` if element with index `state_idx` is fixed.

Implements [teb::VertexType](#).

Definition at line 198 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`.

Referenced by `teb::StateVertex< p >::getDataFree()`, `teb::StateVertex< p >::plusFree()`, and `teb::StateVertex< p >::setFree()`.

**16.36.4.11** `template<int p> StateVertex<p>& teb::StateVertex< p >::operator= ( const StateVertex< p > & rhs )`  
`[inline]`

Definition at line 218 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`, `teb::StateVertex< p >::_states`, `teb::StateVertex< p >::fixed_states()`, and `teb::StateVertex< p >::states()`.

**16.36.4.12** `template<int p> virtual void teb::StateVertex< p >::plus ( const VertexType * rhs )` `[inline],`  
`[virtual]`

Implements [teb::VertexType](#).

Definition at line 84 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`, and `teb::StateVertex< p >::states()`.

**16.36.4.13** `template<int p> virtual void teb::StateVertex< p >::plus ( const double * rhs )` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 91 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`.

**16.36.4.14** `template<int p> virtual void teb::StateVertex< p >::plusFree ( const double * rhs )` `[inline],`  
`[virtual]`

Implements [teb::VertexType](#).

Definition at line 97 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`, and `teb::StateVertex< p >::isFixedComp()`.

**16.36.4.15** `template<int p> virtual void teb::StateVertex< p >::pop ( )` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 203 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_backup`, and `teb::StateVertex< p >::top()`.

16.36.4.16 `template<int p> virtual void teb::StateVertex<p>::push ( ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 202 of file `teb_vertices.h`.

References `teb::StateVertex<p>::_backup`, and `teb::StateVertex<p>::_states`.

16.36.4.17 `template<int p> void teb::StateVertex<p>::setFixedAll ( bool fixed ) [inline]`

Parameters

<i>fixed</i>	set to <code>true</code> in order to fix all <i>p</i> states
--------------	--

Definition at line 158 of file `teb_vertices.h`.

References `teb::StateVertex<p>::_fixed_states`.

16.36.4.18 `template<int p> void teb::StateVertex<p>::setFixedState ( unsigned int state_idx, bool fixed ) [inline]`

Parameters

<i>state_idx</i>	index of the underlying <code>StateVector</code> component that should be fixed.
<i>fixed</i>	set to <code>true</code> in order to fix the selected state.

Definition at line 169 of file `teb_vertices.h`.

References `teb::StateVertex<p>::_fixed_states`.

16.36.4.19 `template<int p> void teb::StateVertex<p>::setFixedStates ( bool fixed ) [inline]`

Parameters

<i>fixed</i>	set to <code>true</code> in order to fix all <i>p</i> states.
--------------	---

Definition at line 175 of file `teb_vertices.h`.

References `teb::StateVertex<p>::_fixed_states`.

16.36.4.20 `template<int p> void teb::StateVertex<p>::setFixedStates ( const bool * fixed ) [inline]`

Parameters

<i>fixed</i>	boolean array with <i>p</i> elements in which each <code>true</code> sets the corresponding state of the <code>StateVector</code> to fixed.
--------------	---

Definition at line 181 of file `teb_vertices.h`.

References `teb::StateVertex<p>::_fixed_states`.

16.36.4.21 `template<int p> void teb::StateVertex<p>::setFixedStates ( const Eigen::Ref< const FixedStates > & fixed_states ) [inline]`

Parameters

<i>fixed_states</i>	Eigen::Vector with <i>p</i> booleans in which each <code>true</code> sets the corresponding state of the State-Vector to fixed.
---------------------	---

Definition at line 187 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_fixed_states`, and `teb::StateVertex< p >::fixed_states()`.

**16.36.4.22** `template<int p> virtual void teb::StateVertex< p >::setFree ( const double * rhs ) [inline],  
[virtual]`

Implements [teb::VertexType](#).

Definition at line 111 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`, and `teb::StateVertex< p >::isFixedComp()`.

**16.36.4.23** `void teb::VertexType::setOptVecIdx ( int opt_vec_idx ) [inline],[inherited]`

Parameters

<i>opt_vec_idx</i>	index of the first value
--------------------	--------------------------

Definition at line 53 of file `graph.h`.

References `teb::VertexType::opt_vec_idx`.

**16.36.4.24** `template<int p> virtual void teb::StateVertex< p >::setStates ( const Eigen::Ref< const StateVector > & s  
) [inline],[virtual]`

Parameters

<i>s</i>	state vector [p x 1]
----------	----------------------

Definition at line 150 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`.

**16.36.4.25** `template<int p> virtual void teb::StateVertex< p >::setStates ( double state ) [inline],[virtual]`

Parameters

<i>state</i>	single state component.
--------------	-------------------------

Definition at line 151 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`.

**16.36.4.26** `template<int p> virtual unsigned int teb::StateVertex< p >::stackSize ( ) const [inline],  
[virtual]`

Implements [teb::VertexType](#).

Definition at line 214 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_backup`.

**16.36.4.27** `template<int p> const StateVector& teb::StateVertex< p >::states ( ) const [inline]`

**Returns**

state vector [p x 1]

Definition at line 148 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`.

Referenced by `teb::EdgeSystemDynamics< p, q, central >::computeValues()`, `teb::EdgeQuadraticForm< p, q >::computeValues()`, `teb::StateVertex< p >::operator=()`, and `teb::StateVertex< p >::plus()`.

**16.36.4.28** `template<int p> StateVector& teb::StateVertex< p >::states ( ) [inline]`

**Returns**

state vector [p x 1]

Definition at line 149 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_states`.

**16.36.4.29** `template<int p> virtual void teb::StateVertex< p >::top ( ) [inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 208 of file `teb_vertices.h`.

References `teb::StateVertex< p >::_backup`, and `teb::StateVertex< p >::_states`.

Referenced by `teb::StateVertex< p >::pop()`.

**16.36.5 Friends And Related Function Documentation**

**16.36.5.1** `template<int p> bool operator!= ( const StateVertex< p > & lhs, const StateVertex< p > & rhs ) [friend]`

Definition at line 235 of file `teb_vertices.h`.

**16.36.5.2** `template<int p> std::ostream& operator<< ( std::ostream & os, const StateVertex< p > & state ) [friend]`

**Parameters**

<i>os</i>	output stream object
<i>state</i>	vector specific <a href="#">StateVertex</a> object

**Returns**

output stream object including a formatted string containing the state vector.

Definition at line 246 of file `teb_vertices.h`.

**16.36.5.3** `template<int p> bool operator== ( const StateVertex< p > & lhs, const StateVertex< p > & rhs ) [friend]`

Definition at line 230 of file `teb_vertices.h`.

### 16.36.6 Member Data Documentation

16.36.6.1 `template<int p> BackupStackType teb::StateVertex< p >::_backup` `[protected]`

Definition at line 254 of file `teb_vertices.h`.

Referenced by `teb::StateVertex< p >::discardTop()`, `teb::StateVertex< p >::pop()`, `teb::StateVertex< p >::push()`, `teb::StateVertex< p >::stackSize()`, and `teb::StateVertex< p >::top()`.

16.36.6.2 `template<int p> int teb::StateVertex< p >::_dimension = p` `[protected]`

Definition at line 256 of file `teb_vertices.h`.

16.36.6.3 `template<int p> FixedStates teb::StateVertex< p >::_fixed_states = FixedStates::Constant(false)`  
`[protected]`

Definition at line 257 of file `teb_vertices.h`.

Referenced by `teb::StateVertex< p >::dimensionFree()`, `teb::StateVertex< p >::fixed_states()`, `teb::StateVertex< p >::isFixedAll()`, `teb::StateVertex< p >::isFixedAny()`, `teb::StateVertex< p >::isFixedComp()`, `teb::StateVertex< p >::operator=()`, `teb::StateVertex< p >::setFixedAll()`, `teb::StateVertex< p >::setFixedState()`, `teb::StateVertex< p >::setFixedStates()`, and `teb::StateVertex< p >::StateVertex()`.

16.36.6.4 `int teb::VertexType::_opt_vec_idx = -1` `[protected]`, `[inherited]`

Definition at line 69 of file `graph.h`.

Referenced by `teb::VertexType::getOptVecIdx()`, and `teb::VertexType::setOptVecIdx()`.

16.36.6.5 `template<int p> StateVector teb::StateVertex< p >::_states` `[protected]`

Definition at line 253 of file `teb_vertices.h`.

Referenced by `teb::StateVertex< p >::getData()`, `teb::StateVertex< p >::getDataFree()`, `teb::StateVertex< p >::operator=()`, `teb::StateVertex< p >::plus()`, `teb::StateVertex< p >::plusFree()`, `teb::StateVertex< p >::push()`, `teb::StateVertex< p >::setFree()`, `teb::StateVertex< p >::setStates()`, `teb::StateVertex< p >::states()`, and `teb::StateVertex< p >::top()`.

16.36.6.6 `template<int p> const int teb::StateVertex< p >::Dimension = p` `[static]`

Definition at line 29 of file `teb_vertices.h`.

16.36.6.7 `template<int p> const int teb::StateVertex< p >::DimStates = p` `[static]`

Definition at line 28 of file `teb_vertices.h`.

The documentation for this class was generated from the following file:

- [teb\\_vertices.h](#)

## 16.37 `teb::SystemDynamics< p, q >` Class Template Reference

Helper class for modeling nonlinear dynamic systems.

```
#include <system_dynamics.h>
```

## Public Types

- using [StateVector](#) = Eigen::Matrix< double, p, 1 >  
*Typedef for state vector with p states [p x 1].*
- using [ControlVector](#) = Eigen::Matrix< double, q, 1 >  
*Typedef for control input vector with q controls [q x 1].*

## Public Member Functions

- [SystemDynamics](#) ()  
*Empty Constructor.*
- virtual [~SystemDynamics](#) ()  
*Empty Destructor.*
- virtual [StateVector stateSpaceModel](#) (const Eigen::Ref< const [StateVector](#) > &x, const Eigen::Ref< const [ControlVector](#) > &u) const =0  
*Implement the nonlinear state space model  $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$  here.*

## Static Public Attributes

- static const int [DimStates](#) = p  
*Store number of states as static variable.*
- static const int [DimControls](#) = q  
*Store number of control inputs as static variable.*

## Protected Member Functions

- [StateVector finiteElemFwdEuler](#) (const Eigen::Ref< const [StateVector](#) > &x1, const Eigen::Ref< const [StateVector](#) > &x2, double dt)  
 *$\dot{x} = stateSpaceModel(x, u)$*
- [StateVector finiteElemCentralDiff](#) (const Eigen::Ref< const [StateVector](#) > &x\_km1, const Eigen::Ref< const [StateVector](#) > &x\_kp1, double dt)  
*Central difference approximation for continuous-time derivatives.*

## Friends

- template<int pf, int qf, bool cf>  
class [EdgeSystemDynamics](#)
- template<int pf, int qf>  
class [Simulator](#)

### 16.37.1 Detailed Description

template<int p, int q>class teb::SystemDynamics< p, q >

This abstract class provides an interface for modeling dynamic systems utilizing nonlinear state space equations:

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

$\mathbf{x} \in \mathbb{R}^p$  denotes the state vector and  $\mathbf{u} \in \mathbb{R}^q$  denotes the control input vector. The system dynamics are modeled in continuous time-domain and discretized by the program itself.



- E.g. the `TebController` class discretizes the system equations using `teb::FiniteDifferences` (see `EdgeSystemDynamics`, `finiteElemFwdEuler()` and `finiteElemCentralDiff()`).
- And the `Simulator` class is able to apply different numerical integration schemes to the state space model implemented here (see `NumericalIntegrator`).

**Todo** Methods/Interface for jacobian and hessian calculation

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### Template Parameters

$p$	Number of state variables
$q$	Number of input variables

#### Examples:

[integrator\\_system1.cpp](#), [integrator\\_system2.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [integrator\\_system\\_mex.cpp](#), [integrator\\_system\\_sfuns.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [linear\\_system\\_state\\_space.cpp](#), [mobile\\_robot\\_teb.cpp](#), [rocket\\_system.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Definition at line 34 of file `system_dynamics.h`.

### 16.37.2 Member Typedef Documentation

16.37.2.1 `template<int p, int q> using teb::SystemDynamics< p, q >::ControlVector = Eigen::Matrix<double, q, 1>`

Definition at line 50 of file `system_dynamics.h`.

16.37.2.2 `template<int p, int q> using teb::SystemDynamics< p, q >::StateVector = Eigen::Matrix<double, p, 1>`

Definition at line 48 of file `system_dynamics.h`.

### 16.37.3 Constructor & Destructor Documentation

16.37.3.1 `template<int p, int q> teb::SystemDynamics< p, q >::SystemDynamics ( ) [inline]`

Definition at line 52 of file `system_dynamics.h`.

16.37.3.2 `template<int p, int q> virtual teb::SystemDynamics< p, q >::~~SystemDynamics ( ) [inline], [virtual]`

Definition at line 53 of file `system_dynamics.h`.

### 16.37.4 Member Function Documentation

16.37.4.1 `template<int p, int q> StateVector teb::SystemDynamics< p, q >::finiteElemCentralDiff ( const Eigen::Ref< const StateVector > & x_km1, const Eigen::Ref< const StateVector > & x_kp1, double dt ) [inline], [protected]`

Approximate derivatives using  $\dot{x}(t) = \frac{x_{k+1} - x_{k-1}}{2\Delta T}$

## Parameters

$x_{km1}$	Discrete state vector $\mathbf{x}_{k-1}$ [p x 1]
$x_{kp1}$	Discrete state vector $\mathbf{x}_{k+1}$ [p x 1]
$dt$	Time interval / discretization step width $\Delta T$

## Returns

Finite difference approximation

Definition at line 107 of file system\_dynamics.h.

**16.37.4.2** `template<int p, int q> StateVector teb::SystemDynamics< p, q >::finiteElemFwdEuler ( const Eigen::Ref< const StateVector > & x1, const Eigen::Ref< const StateVector > & x2, double dt ) [inline], [protected]`

Forward difference approximation for continuous-time derivatives

Approximate derivatives using  $\dot{x}(t) = \frac{x_{k+1} - x_k}{\Delta T}$

## Parameters

$x1$	Discrete state vector $\mathbf{x}_k$ [p x 1]
$x2$	Discrete state vector $\mathbf{x}_{k+1}$ [p x 1]
$dt$	Time interval / discretization step width $\Delta T$

## Returns

Finite difference approximation

Definition at line 91 of file system\_dynamics.h.

**16.37.4.3** `template<int p, int q> virtual StateVector teb::SystemDynamics< p, q >::stateSpaceModel ( const Eigen::Ref< const StateVector > & x, const Eigen::Ref< const ControlVector > & u ) const [pure virtual]`

This method defines the state space model of the dedicated dynamic system.

Implement only the right hand side  $f(\cdot)$  of  $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$  and return the equation as a StateVector type.

E.g.  $\dot{x} = x + u$   $x, u \in \mathbb{R}$  (with  $p = 1$  and  $q = 1$ )

```
StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
    Eigen::Ref<const ControlVector>& u) const
{
    StateVector f = x[0] + u[0];
    return f;
}
```

## Parameters

$x$	State vector $\mathbf{x}$ [p x 1]
$u$	Control input vector $\mathbf{u}$ [q x 1]

## Returns

$f(\mathbf{x}, \mathbf{u})$  as vector [p x 1]

## Examples:

[integrator\\_system1.cpp](#), [integrator\\_system2.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [integrator\\_system\\_mex.cpp](#), [integrator\\_system\\_sfuns.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [linear\\_system\\_state\\_space.cpp](#), [mobile\\_robot\\_teb.cpp](#), [rocket\\_system.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Referenced by `teb::ExplicitEuler< p, q >::integrate()`, `teb::RungeKuttaClassic< p, q >::integrate()`, and `teb::RungeKutta5thOrder< p, q >::integrate()`.

### 16.37.5 Friends And Related Function Documentation

16.37.5.1 `template<int p, int q> template<int pf, int qf, bool cf> friend class EdgeSystemDynamics` `[friend]`

Definition at line 37 of file `system_dynamics.h`.

16.37.5.2 `template<int p, int q> template<int pf, int qf> friend class Simulator` `[friend]`

Definition at line 40 of file `system_dynamics.h`.

### 16.37.6 Member Data Documentation

16.37.6.1 `template<int p, int q> const int teb::SystemDynamics< p, q >::DimControls = q` `[static]`

Definition at line 45 of file `system_dynamics.h`.

16.37.6.2 `template<int p, int q> const int teb::SystemDynamics< p, q >::DimStates = p` `[static]`

Definition at line 44 of file `system_dynamics.h`.

The documentation for this class was generated from the following file:

- [system\\_dynamics.h](#)

## 16.38 `teb::Config::Teb` Struct Reference

Configurations related to the TEB algorithm.

```
#include <config.h>
```

### Public Attributes

- unsigned int `n_min` = 3  
*Minimum number of samples (states and control input pairs) allowed.*
- unsigned int `n_pre` = 20  
*Initial number of samples (used in [TebController::updateGoal\(\)](#)).*
- unsigned int `n_max` = 150  
*Maximum number of samples allowed.*
- double `dt_min` = 0.01  
*Minimum dt (only considered if `EdgePositiveTime` is activated).*
- double `dt_ref` = 0.1  
*Desired reference sampling time (the TEB tries to converge towards `dt_ref`) (make sure `dt_ref`>0).*
- double `dt_hyst` = 0.01  
*Sampling time hysteresis used in `resizeTEB` to avoid oscillations (default: `0.1*dt_ref`).*
- double `goal_dist_force_reinit` = 0.5  
*Force reinit of the trajectory if distance between current and new goal is above threshold.*
- unsigned int `teb_iter` = 5  
*Number of outer-loop TEB iterations (used for guiding  $\Delta T$  towards `dt_ref`, see [TebController::optimizeTEB\(\)](#)).*

- `FiniteDifferences diff_method = FiniteDifferences::FORWARD`

*Select finite difference method for discretizing system dynamics and obtaining  $\Delta T$ .*

### 16.38.1 Detailed Description

Definition at line 60 of file config.h.

### 16.38.2 Member Data Documentation

#### 16.38.2.1 `FiniteDifferences teb::Config::Teb::diff_method = FiniteDifferences::FORWARD`

Definition at line 70 of file config.h.

#### 16.38.2.2 `double teb::Config::Teb::dt_hyst = 0.01`

Definition at line 67 of file config.h.

#### 16.38.2.3 `double teb::Config::Teb::dt_min = 0.01`

Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 65 of file config.h.

Referenced by `teb::TebController< p, q >::setupHorizon()`.

#### 16.38.2.4 `double teb::Config::Teb::dt_ref = 0.1`

Definition at line 66 of file config.h.

Referenced by `teb::TebController< p, q >::resetController()`, and `teb::TebController< p, q >::setupHorizon()`.

#### 16.38.2.5 `double teb::Config::Teb::goal_dist_force_reinit = 0.5`

Definition at line 68 of file config.h.

#### 16.38.2.6 `unsigned int teb::Config::Teb::n_max = 150`

Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 64 of file config.h.

#### 16.38.2.7 `unsigned int teb::Config::Teb::n_min = 3`

Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 62 of file config.h.

Referenced by `teb::TebController< p, q >::setupHorizon()`.

16.38.2.8 `unsigned int teb::Config::Teb::n_pre = 20`

Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 63 of file `config.h`.

Referenced by `teb::TebController< p, q >::setupHorizon()`.

16.38.2.9 `unsigned int teb::Config::Teb::teb_iter = 5`

Definition at line 69 of file `config.h`.

Referenced by `teb::BaseSolverLeastSquares::adaptWeights()`.

The documentation for this struct was generated from the following file:

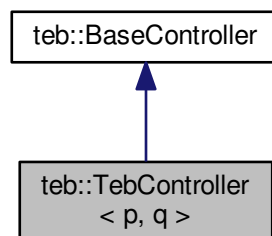
- [config.h](#)

## 16.39 `teb::TebController< p, q >` Class Template Reference

Main Timed-Elastic-Band controller class.

`#include <teb_controller.h>`

Inheritance diagram for `teb::TebController< p, q >`:



### Public Types

- using `StateVector` = `Eigen::Matrix< double, p, 1 >`  
*Typedef for state vector with p states [p x 1].*
- using `ControlVector` = `Eigen::Matrix< double, q, 1 >`  
*Typedef for control input vector with q controls [q x 1].*
- using `StateSequence` = `std::deque< StateVertex< p > >`  
*Typedef for the state sequence that contains all state vertices.*
- using `ControlSequence` = `std::deque< ControlVertex< q > >`  
*Typedef for the control sequence that contains all control vertices.*
- using `EdgeContainer` = `typename EdgeType::EdgeContainer`  
*Typedef for edge containers that stores all edges required for the optimization (Hyper-Graph).*

- using `VertexContainer` = typename `VertexType::VertexContainer`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*
- using `FixedStates` = `Eigen::Matrix< bool, p, 1 >`  
*Typedef to represent a fixed/unfixed boolean vector w.r.t. states (StateVector)  $[p \times 1]$ .*

## Public Member Functions

- `TebController` (`BaseSolver` \*solver=nullptr)  
*Default constructor.*
- `TebController` (const `Config` \*config, `BaseSolver` \*solver=nullptr)  
*Extended constructor with config pointer.*
- virtual `~TebController` ()  
*!*
- virtual `Eigen::VectorXd` `firstState` () const  
*Access first state of the TEB sequence.*
- const `StateVector` & `firstStateRef` () const  
*Access first state of the TEB sequence.*
- virtual `Eigen::VectorXd` `lastState` () const  
*Access last state of the TEB sequence.*
- const `StateVector` & `lastStateRef` () const  
*Access last state of the TEB sequence.*
- virtual `Eigen::VectorXd` `firstControl` () const  
*Access first control input of the TEB sequence (used for MPC)*
- const `ControlVector` & `firstControlRef` () const  
*Access first control input of the TEB sequence (used for MPC)*
- `Eigen::VectorXd` `firstControlSaturated` () const  
*Access saturated first control inputs of the TEB sequence (used for MPC)*
- const `StateSequence` & `stateSequence` () const  
*Access complete state sequence container.*
- const `ControlSequence` & `controlSequence` () const  
*Access complete control input sequence container.*
- const `TimeDiff` & `dt` () const  
*Access vertex that stores the TEB time difference  $\Delta T$ .*
- virtual double `getDt` () const  
*Return the TEB time difference  $\Delta T$ .*
- virtual unsigned int `getN` () const  
*Get current number of discrete state vectors in the state sequence.*
- virtual unsigned int `getM` () const  
*Get current number of discrete control vectors in the control input sequence.*
- virtual `Eigen::MatrixXd` `returnControlInputSequence` () const  
*Return the complete control input sequence  $u_k, k = 1, \dots, n - 1$ .*
- const `HyperGraph` & `graph` () const  
*Access optimizable hyper-graph instance (read-only).*
- virtual void `initTrajectory` (const `Eigen::Ref< const StateVector >` &x0, const `Eigen::Ref< const StateVector >` &x1, unsigned int n)  
*Initialize trajectory between start  $x_0$  and goal  $x_f$  with n discrete samples.*
- void `resizeTrajectory` ()  
*Resize existing trajectory according to current sample time  $\Delta T$ .*
- void `resampleTrajectory` (unsigned int n\_new)  
*Resample trajectory using linear interpolation.*

- virtual void [updateStart](#) (const Eigen::Ref< const [StateVector](#) > &x0)  
*Set new start state.*
- virtual void [updateGoal](#) (const Eigen::Ref< const [StateVector](#) > &xf)  
*Set new final state.*
- void [removeStateControlInputPair](#) (unsigned int sample\_no, bool predict\_control=true)  
*Remove StateVector and ControlVector at index `sample_no` from the state and control sequences.*
- void [insertStateControlInputPair](#) (unsigned int sample\_idx, const Eigen::Ref< const [StateVector](#) > &state, const Eigen::Ref< const [ControlVector](#) > &control, bool predict\_control=true)  
*Insert StateVector and ControlVector to the state and control sequences at index `sample_idx`.*
- void [predictControl](#) (Eigen::Ref< [ControlVector](#) > ctrl\_out, const Eigen::Ref< const [StateVector](#) > &x1, const Eigen::Ref< const [StateVector](#) > &x2)  
*Predict control  $u$  that corresponds to the transition from  $x_1$  to  $x_2$ .*
- void [pushBackStateControlInputPair](#) (const Eigen::Ref< const [StateVector](#) > &state, const Eigen::Ref< const [ControlVector](#) > &control)  
*Add a state vector  $x_k$  and a control input  $u_k$  to the end of the state and control sequences.*
- void [pushFrontStateControlInputPair](#) (const Eigen::Ref< const [StateVector](#) > &state, const Eigen::Ref< const [ControlVector](#) > &control)  
*Add a state vector  $x_k$  and a control input  $u_k$  to the beginning of the state and control sequences.*
- void [setGoalStatesFixedOrUnfixed](#) (const bool \*fixed\_goal\_states)  
*Set individual goal state vector components fixed or unfixed for optimization.*
- void [setGoalStatesFixedOrUnfixed](#) (const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> &fixed\_goal\_states)  
*Set individual goal state vector components fixed or unfixed for optimization (Eigen version)*
- void [setGoalStatesFixedOrUnfixed](#) (bool all\_goal\_states\_fixed)  
*Set all goal state vector components fixed or unfixed for optimization.*
- void [setupHorizon](#) (bool fixed\_goal, bool fixed\_resolution, int no\_samples=-1, double dt=-1)  
*Setup horizon for the underlying optimal control problem.*
- virtual void [resetController](#) ()  
*Delete all Vertices (State and control input sequences), all Edges (Hyper-Graph, `clearEdges()`) and reset  $\Delta T$  to `Config::Teb::dt_ref`.*
- virtual void [buildOptimizationGraph](#) ()  
*Build Hyper-Graph for graph based optimization.*
- virtual void [customOptimizationGraph](#) ()  
*Override to add or modify the hyper-graph created in `buildOptimizationGraph()`*
- virtual void [customOptimizationGraphHotStart](#) ()  
*Use this method to apply some changes before hot-starting from previous graph structures.*
- void [initOptimization](#) ()  
*Initialize optimization - run before calling `optimizeTEB()`.*
- void [optimizeTEB](#) ()  
*TEB optimization function (includes outer loop and `BaseSolver` calls).*
- virtual void [step](#) (const double \*const x0, const double \*const xf, double \*ctrl\_out=nullptr)  
*Perform complete MPC step (initialization if necessary or update and optimization)*
- void [step](#) (const Eigen::Ref< const [StateVector](#) > &x0, const Eigen::Ref< const [StateVector](#) > &xf, Eigen::Map< [ControlVector](#) > \*ctrl\_out=nullptr)  
*Perform complete MPC step (initialization if necessary or update and optimization) (Eigen version)*
- void [activateObjectiveTimeOptimal](#) (bool activate=true)  
*Activate predefined edge for time-optimal control.*
- void [activateObjectiveQuadraticForm](#) (double Q, double R, double Qf, bool activate=true)  
*Activate predefined edge for minimizing the quadratic form.*
- void [activateControlBounds](#) (const Eigen::Ref< const [ControlVector](#) > u\_min, const Eigen::Ref< const [ControlVector](#) > u\_max, bool activate=true)  
*Activate predefined edge for control input bounds.*

- void [activateControlBounds](#) (unsigned int control\_idx, double u\_min, double u\_max, bool activate=true)  
*Activate predefined edge for control input bounds (single component version)*
- void [activateStateBounds](#) (const Eigen::Ref< const [StateVector](#) > &x\_min, const Eigen::Ref< const [StateVector](#) > &x\_max, bool activate=true)  
*Activate predefined edge for state bounds.*
- void [activateStateBounds](#) (unsigned int state\_idx, double x\_min, double x\_max, bool activate=true)  
*Activate predefined edge for state bounds (single component version)*
- void [setFixedDt](#) (bool fixed)  
*Fix the [TimeDiff](#)  $\Delta T$  during optimization.*
- void [setSystemDynamics](#) ([SystemDynamics](#)< p, q > \*system, bool activate=true)  
*Set system dynamics and activate the predefined equality-constraint-edge.*
- void [setSolver](#) ([BaseSolver](#) \*solver)  
*Set algorithm to solve the underlying optimization problem.*
- Eigen::VectorXd [getAbsoluteTimeVec](#) () const  
*Get vector of absolute times  $t = [0, \Delta T, 2\Delta T, \dots, n\Delta T]$ .*
- Eigen::MatrixXd [getStateCtrlInfoMat](#) () const  
*Get TEB states and control inputs in matrix form (Use for debugging).*
- Eigen::Matrix< bool, p+q,-1 > [getTEBFixedMap](#) () const  
*Get TEB matrix as bool matrix: *true* : fixed, *false* = free (Use for debugging).*
- Eigen::Matrix< double,-1, 2 > [getSampleOptVecIdxVMat](#) () const  
*Get Hessian index (col 1) and number of free variables (col 2) for each [VertexType](#) ([StateVertex](#), [ControlVertex](#), [TimeDiff](#)) (rowwise).*

## Public Attributes

- const [Config](#) \* [cfg](#)  
*Store pointer to the [Config](#) class.*

## Static Public Attributes

- static const unsigned int [NoStates](#) = p  
*Store number of states as static variable.*
- static const unsigned int [NoControls](#) = q  
*Store number of control inputs as static variable.*

## Protected Member Functions

- void [setFixedStart](#) (bool fixed)  
*Fix the complete first [StateVector](#) of the state sequence (start state vector).*
- void [setFixedStart](#) (const bool \*fixed)  
*Fix selected components of the first [StateVector](#) of the state sequence (start state vector).*
- void [setFixedGoal](#) (bool fixed)  
*Fix the complete last [StateVector](#) of the state sequence (final/goal state vector).*
- void [setFixedGoal](#) (const bool \*fixed)  
*Fix selected components of the last [StateVector](#) of the state sequence (final state vector).*
- void [setFixedGoal](#) (const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> &fixed)  
*Fix selected components of the last [StateVector](#) of the state sequence (final state vector).*
- void [getActiveVertices](#) ()  
*Collect active vertices and determine the Hessian start index for each vertex ([StateVertex](#), [ControlVertex](#) and [TimeDiff](#)).*



## Protected Attributes

- [StateSequence \\_state\\_seq](#)  
Store the complete sequence of state vectors (represented as [StateVertex](#) for the representation in a hyper-graph)
- [ControlSequence \\_ctrl\\_seq](#)  
Store the complete sequence of control input vectors (represented as [ControlVertex](#) for the representation in a hyper-graph)
- [TimeDiff \\_dt](#)  
[TimeDiff](#) vertex that represents the required transition time between consecutive states and control inputs.
- [StateVector \\_goal\\_backup](#)  
Store and backup goal vector (should be changed in all functions that set the goal point).
- `int _no_samples = -1`  
Number of samples for trajectory initialization (if -1, [Config::Teb::n\\_pre](#) is used, see [setupHorizon\(\)](#)).
- `double _dt_ref = -1`  
Reference time difference (discretization) (if -1, [Config::Teb::dt\\_ref](#) is used, see [setupHorizon\(\)](#)).
- `bool _optimized`  
Status flag that is `true`, if at least one optimization step in [optimizeTEB\(\)](#) is performed before.
- [HyperGraph \\_graph](#)  
Instance of the optimization hyper-graph to store all active vertices and edges (interface for the solver)
- [FixedStates \\_fixed\\_goal\\_states](#) = `FixedStates::Constant(true)`  
Store information about (un-)fixed components of the goal [StateVector](#) (initialized to fixed, see [buildOptimizationGraph\(\)](#)).
- [BaseSolver \\* \\_solver](#) = `nullptr`  
Pointer to solver class specified using [setSolver\(\)](#) or [TebController\(\)](#).
- `bool _cfg_owned = false`  
If `true`, the [Config TebController::cfg](#) is owned by this class, that requires its deletion inside `~TebController()`.
- `bool _active_time_optimal = false`  
if `true`, [EdgeMinimizeTime](#) is activated (see [activateObjectiveTimeOptimal\(\)](#)).
- `std::tuple< bool, double, double, double > _active_quadratic_form = std::make_tuple(false,1,1,1)`  
Stores activation status of [EdgeQuadraticForm](#).
- `std::pair< bool, SystemDynamics< p, q > * > _active_system_dynamics = std::make_pair(false,nullptr)`  
Stores activation status of [EdgeSystemDynamics](#) and a pointer to the problem specific [SystemDynamics](#) object.
- `std::tuple< bool, ControlVector, ControlVector > _active_control_bounds = std::make_tuple(false,ControlVector::Constant(INF),ControlVector::Constant(-INF))`  
Stores activation status of [EdgeControlBounds](#) and minimum/maximum bounds on each [ControlVector](#).
- `std::tuple< bool, StateVector, StateVector > _active_state_bounds = std::make_tuple(false, StateVector::Constant(-INF), StateVector::Constant(INF))`  
Stores activation status of [EdgeStateBounds](#) and minimum/maximum bounds on each [StateVector](#).

## 16.39.1 Detailed Description

```
template<int p, int q>class teb::TebController< p, q >
```

This class defines the main TEB controller functions required to controll a dynamic system.

The underlying optimization problem is defined as a hyper-graph in which edges define cost/objective functions (see [TebController::buildOptimizationGraph](#) and [TebController::customOptimizationGraph](#) class methods).

To overcome the creation of custom hyper-graphs, one can use the [SystemDynamics](#) class and pass the system to this class using [TebController::setSystemDynamics](#). Other predefined edges are listed below.

The TEB controller requires a dedicated solver (inherited from [BaseSolver](#)) that can be passed using the `setSolver` method.

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

**Todo** Using the [TebController](#) inside a standard container (e.g. vector) does not work as supposed to do.  
Create copy and move constructors.

#### See Also

[TebController::buildOptimizationGraph](#), [TebController::customOptimizationGraph](#), [SystemDynamics](#), [TebController::setSystemDynamics](#), [TebController::activateObjectiveTimeOptimal](#), [TebController::activateControlBounds](#), [TebController::activateStateBounds](#), [TebController::setSolver](#) [BaseSolver](#)

#### Template Parameters

$p$	Number of states / Size of the state vector
$q$	Number of control inputs / Size of the control vectorp: number of states, q: number of control inputs

#### Examples:

[integrator\\_system1.cpp](#), [integrator\\_system2.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [integrator\\_system\\_mex.cpp](#), [integrator\\_system\\_sfuns.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#), [linear\\_system\\_state\\_space.cpp](#), [rocket\\_system.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Definition at line 54 of file `teb_controller.h`.

### 16.39.2 Member Typedef Documentation

16.39.2.1 `template<int p, int q> using teb::TebController< p, q >::ControlSequence = std::deque< ControlVertex<q> >`

Definition at line 71 of file `teb_controller.h`.

16.39.2.2 `template<int p, int q> using teb::TebController< p, q >::ControlVector = Eigen::Matrix<double, q, 1>`

Definition at line 67 of file `teb_controller.h`.

16.39.2.3 `template<int p, int q> using teb::TebController< p, q >::EdgeContainer = typename EdgeType::EdgeContainer`

Definition at line 73 of file `teb_controller.h`.

16.39.2.4 `template<int p, int q> using teb::TebController< p, q >::FixedStates = Eigen::Matrix<bool,p,1>`

Definition at line 77 of file `teb_controller.h`.

16.39.2.5 `template<int p, int q> using teb::TebController< p, q >::StateSequence = std::deque< StateVertex<p> >`

Definition at line 69 of file `teb_controller.h`.

16.39.2.6 `template<int p, int q> using teb::TebController< p, q >::StateVector = Eigen::Matrix<double,p,1>`

Definition at line 65 of file teb\_controller.h.

16.39.2.7 `template<int p, int q> using teb::TebController< p, q >::VertexContainer = typename  
VertexType::VertexContainer`

Definition at line 75 of file teb\_controller.h.

### 16.39.3 Constructor & Destructor Documentation

16.39.3.1 `template<int p, int q> teb::TebController< p, q >::TebController ( BaseSolver * solver = nullptr )  
[inline]`

This constructor creates a default [Config](#) object for TEB and solver settings.

Parameters

<i>solver</i>	Register solver type as subclass of <a href="#">BaseSolver</a> . If not passed, you must use <a href="#">setSolver()</a> later.
---------------	---

Definition at line 101 of file teb\_controller.h.

References `teb::TebController< p, q >::_cfg_owned`.

16.39.3.2 `template<int p, int q> teb::TebController< p, q >::TebController ( const Config * config, BaseSolver *  
solver = nullptr ) [inline]`

Pass config class to controller and solver. [TebController](#) does not free [Config](#) afterwards.

Parameters

<i>config</i>	Set user-defined config for the controller and solver.
<i>solver</i>	Register solver type as subclass of <a href="#">BaseSolver</a> . If not passed, you must use <a href="#">setSolver()</a> later.

Definition at line 111 of file teb\_controller.h.

References `teb::TebController< p, q >::setSolver()`.

16.39.3.3 `template<int p, int q> virtual teb::TebController< p, q >::~~TebController ( ) [inline],  
[virtual]`

Destructor. Memory is freed if necessary.

Definition at line 119 of file teb\_controller.h.

References `teb::TebController< p, q >::_cfg_owned`, `teb::TebController< p, q >::_graph`, `teb::TebController< p, q >::cfg`, and `teb::HyperGraph::clearGraph()`.

### 16.39.4 Member Function Documentation

16.39.4.1 `template<int p, int q> void teb::TebController< p, q >::activateControlBounds ( const Eigen::Ref< const  
ControlVector > u_min, const Eigen::Ref< const ControlVector > u_max, bool activate = true )  
[inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{u}_k \leq \mathbf{u}_{max}$
- $\mathbf{u}_k \geq \mathbf{u}_{min}$

## Parameters

<i>u_min</i>	minimum bound on the control input $\mathbf{u}_k$
<i>u_max</i>	maximum bound on the control input $\mathbf{u}_k$
<i>activate</i>	if set to <code>true</code> than the edge is activated.

## See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveTimeOptimal\(\)](#), [activateStateBounds\(\)](#), [setSystemDynamics\(\)](#)

Definition at line 481 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_control_bounds`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.2** `template<int p, int q> void teb::TebController< p, q >::activateControlBounds ( unsigned int control_idx, double u_min, double u_max, bool activate = true ) [inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{u}_k \leq \mathbf{u}_{max}$  (component-wise)
- $\mathbf{u}_k \geq \mathbf{u}_{min}$  (component-wise)

The default initialization of the bounds is  $-\text{INF}$  and  $\text{INF}$  respectively, if this predefined edge is activated. This version of [activateControlBounds\(\)](#) makes use of the default initialization and just updates the bounds for index `state_index`. All other bounds remain untouched (and if not changed before, they are bound to  $-\text{INF}$  and  $\text{INF}$ ).

## Parameters

<i>control_idx</i>	selected component of $\mathbf{x}_k$ that should be bounded to <code>x_min</code> and <code>x_max</code> .
<i>u_min</i>	minimum bound on the component with index <code>control_idx</code> of control input $\mathbf{u}_k$ .
<i>u_max</i>	maximum bound on the component with index <code>control_idx</code> of the control input $\mathbf{u}_k$ .
<i>activate</i>	if set to <code>true</code> than the edge is activated.

**Todo** Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

## See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveTimeOptimal\(\)](#), [activateControlBounds\(\)](#), [setSystemDynamics\(\)](#)

Definition at line 510 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_control_bounds`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.3** `template<int p, int q> void teb::TebController< p, q >::activateObjectiveQuadraticForm ( double Q, double R, double Qf, bool activate = true ) [inline]`

Adds the following cost function

$$J = (\mathbf{x}_n - \mathbf{x}_f)^T \mathbf{Q}_f (\mathbf{x}_n - \mathbf{x}_f) + \sum_{k=0}^{n-1} ((\mathbf{x}_k - \mathbf{x}_f)^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_f) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k)$$

to the global cost/objective function.

The weight matrices are assumed to be diagonal matrices with unifrom values  $Q$ ,  $R$  and  $Qf$ .

See [EdgeQuadraticForm](#) for further details.

## Parameters

$Q$	Uniform scalar weight for minimizing the squared state error
$R$	Uniform scalar weight for minimizing the squared control input
$Q_f$	Uniform scalar weight for minimizing the squared state error for the final state (last state of the horizon)
<i>activate</i>	if set to <code>true</code> than the edge is activated.

## See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveTimeOptimal\(\)](#), [activateControlBounds\(\)](#), [activateStateBounds\(\)](#), [setSystemDynamics\(\)](#)

Definition at line 460 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_quadratic_form`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.4** `template<int p, int q> void teb::TebController< p, q >::activateObjectiveTimeOptimal ( bool activate = true )`  
`[inline]`

Adds the following cost function  $V(\mathcal{B}) = (n-1)\Delta T$  to the global cost/objective function.

Additionally, the inequality constraint  $\Delta T > \Delta T_{min}$  is added.  $\Delta T_{min}$  can be changed in `Config::Teb::teb_min`.

## Parameters

<i>activate</i>	if set to <code>true</code> than the edge is activated.
-----------------	---

## See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveQuadraticForm\(\)](#), [activateControlBounds\(\)](#), [activateStateBounds\(\)](#), [setSystemDynamics\(\)](#)

Definition at line 435 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_time_optimal`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.5** `template<int p, int q> void teb::TebController< p, q >::activateStateBounds ( const Eigen::Ref< const StateVector > & x_min, const Eigen::Ref< const StateVector > & x_max, bool activate = true )`  
`[inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{x}_k \leq \mathbf{x}_{max}$  (component-wise)
- $\mathbf{x}_k \geq \mathbf{x}_{min}$  (component-wise)

To just bound a subset of components of the state  $\mathbf{x}_k$ , set unbounded components to  $-\text{Inf} \leq x_k \leq \text{Inf}$  using the `INF` macro. Or see overloaded function [activateStateBounds\(\)](#).

## Parameters

<i>x_min</i>	minimum bounds on the state $\mathbf{x}_k$ [p x 1].
<i>x_max</i>	maximum bounds on the state $\mathbf{x}_k$ [p x 1].

<i>activate</i>	if set to <code>true</code> than the edge is activated.
-----------------	---

**Todo** Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveTimeOptimal\(\)](#), [activateControlBounds\(\)](#), [setSystemDynamics\(\)](#)

Definition at line 539 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_state_bounds`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.6** `template<int p, int q> void teb::TebController< p, q >::activateStateBounds ( unsigned int state_idx, double x_min, double x_max, bool activate = true ) [inline]`

Adds the following inequality constraints to the optimization problem:

- $\mathbf{x}_k \leq \mathbf{x}_{max}$  (component-wise)
- $\mathbf{x}_k \geq \mathbf{x}_{min}$  (component-wise)

The default initialization of the bounds is `-INF` and `INF` respectively, if this predefined edge is activated. This version of [activateStateBounds\(\)](#) makes use of the default initialization and just updates the bounds for index `state_idx`. All other bounds remain untouched (and if not changed before, they are bound to `-INF` and `INF`).

Parameters

<i>state_idx</i>	selected component of $\mathbf{x}_k$ that should be bounded to <code>x_min</code> and <code>x_max</code> .
<i>x_min</i>	minimum bound on the component with index <code>state_idx</code> of state $\mathbf{x}_k$ .
<i>x_max</i>	maximum bound on the component with index <code>state_idx</code> of the state $\mathbf{x}_k$ .
<i>activate</i>	if set to <code>true</code> than the edge is activated.

**Todo** Avoid unnecessary calculations for unbounded values (Jacobian, Hessian, ...). Maybe change edge-dimension dynamically in that case.

See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveTimeOptimal\(\)](#), [activateControlBounds\(\)](#), [setSystemDynamics\(\)](#)

Definition at line 568 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_state_bounds`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.7** `template<int p, int q> void teb::TebController< p, q >::buildOptimizationGraph ( ) [virtual]`

Construct the hyper-graph for the underlying optimization problem.

Instantiate edges as subclasses of [BaseEdge](#) and add them to the corresponding containers.

- Add objective/cost function edges to `TebController::_graph` via [HyperGraph::addEdgeObjective\(\)](#).
- equality constraints to `TebController::_graph` via [HyperGraph::addEdgeEquality\(\)](#).
- and inequality constraints to `TebController::_graph` via [HyperGraph::addEdgeInequality\(\)](#).

Example for the objective [EdgeMinimizeTime](#) that is derived from [BaseEdge](#):

```
EdgeMinimizeTime* to_edge = new EdgeMinimizeTime; // Instantiate new edge object.
to_edge->setVertex(&_dt); // Connect edge with vertex TimeDiff.
to_edge->setData(getN()-1); // Set weight/data (only for EdgeMinimizeTime).
_graph.addEdgeObjective(to_edge); // Store pointer to the container.
```

The implementation contains a few default edges commonly used in MPC and Optimal Control scenarios. See:

- [activateObjectiveTimeOptimal\(\)](#)
- [activateObjectiveQuadraticForm\(\)](#)
- [activateControlBounds\(\)](#)
- [activateStateBounds\(\)](#)
- [setSystemDynamics\(\)](#)

After checking and adding activated default edges, the function calls [customOptimizationGraph\(\)](#) for extended user supplied edges.

This method is called within [initOptimization\(\)](#).

#### See Also

[customOptimizationGraph](#), [customOptimizationGraphHotStart](#), [HyperGraph](#), [BaseEdge](#)

**Todo** Here we need to fix the last control input since it undefined (no control is needed to go to state n+1). Maybe we can change the code somewhere else, that the control sequence is always [getN\(\)-1](#).

Definition at line 590 of file `teb_controller.hpp`.

References `teb::FORWARD`, `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::setBounds()`, `teb::EdgeMinimizeTime::setData()`, `teb::EdgeQuadraticForm< p, q >::setReference()`, and `teb::EdgeQuadraticForm< p, q >::setWeights()`.

**16.39.4.8** `template<int p, int q> const ControlSequence& teb::TebController< p, q >::controlSequence ( ) const`  
[inline]

#### Returns

Reference to the control input sequence (read-only)

Definition at line 186 of file `teb_controller.h`.

References `teb::TebController< p, q >::_ctrl_seq`.

**16.39.4.9** `template<int p, int q> virtual void teb::TebController< p, q >::customOptimizationGraph ( )` [inline],  
[virtual]

Derive TEB class and override [customOptimizationGraph\(\)](#) to manually construct the hyper-graph or add edges in addition to activated common edges in [buildOptimizationGraph\(\)](#). See the description of [buildOptimizationGraph\(\)](#) for more information on how to add edges.

#### See Also

[buildOptimizationGraph](#), [customOptimizationGraphHotStart](#), [HyperGraph](#), [BaseEdge](#), [initOptimization\(\)](#)

Definition at line 375 of file `teb_controller.h`.

16.39.4.10 `template<int p, int q> virtual void teb::TebController< p, q >::customOptimizationGraphHotStart ( )`  
`[inline], [virtual]`

Depending on [HyperGraph::isGraphModified\(\)](#) the graph is either cleared and constructed again or the previous graph is used. In the latter case [customOptimizationGraph\(\)](#), [buildOptimizationGraph\(\)](#) and [getActiveVertices\(\)](#) are not called. If you wish to apply changes without recreating the structure add them to this function. It is called inside [initOptimization\(\)](#).

#### See Also

[customOptimizationGraph](#), [buildOptimizationGraph](#), [HyperGraph](#), [BaseEdge](#), [initOptimization\(\)](#)

Definition at line 386 of file `teb_controller.h`.

16.39.4.11 `template<int p, int q> const TimeDiff& teb::TebController< p, q >::dt ( ) const` `[inline]`

#### Returns

Reference to time-diff vertex (read-only)

Definition at line 191 of file `teb_controller.h`.

References `teb::TebController< p, q >::_dt`.

Referenced by `teb::TebController< p, q >::setupHorizon()`.

16.39.4.12 `template<int p, int q> virtual Eigen::VectorXd teb::TebController< p, q >::firstControl ( ) const`  
`[inline], [virtual]`

#### Returns

Copy of the first control input (read-only)

Implements [teb::BaseController](#).

Definition at line 150 of file `teb_controller.h`.

References `teb::TebController< p, q >::_ctrl_seq`.

Referenced by `teb::TebController< p, q >::firstControlSaturated()`, and `teb::TebController< p, q >::step()`.

16.39.4.13 `template<int p, int q> const ControlVector& teb::TebController< p, q >::firstControlRef ( ) const`  
`[inline]`

#### Returns

Reference to the first control input (read-only)

Definition at line 155 of file `teb_controller.h`.

References `teb::TebController< p, q >::_ctrl_seq`.

16.39.4.14 `template<int p, int q> Eigen::VectorXd teb::TebController< p, q >::firstControlSaturated ( ) const`  
`[inline]`

This function saturates / bounds the returned control input  $\mathbf{u}$  to  $[\mathbf{u}_{min}, \mathbf{u}_{max}]$ .

The saturation could be helpful in case of soft constraints, that are violated for the planning and prediction, but that should be satisfied for the real system.

Specify the bounds via [activateControlBounds\(\)](#).



**Todo** Support multiple control input bounds (now its scalar for all elements of **u**).

#### Returns

Saturated control input  $u_k$  [ $q \times 1$ ].

Definition at line 168 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_control_bounds`, and `teb::TebController< p, q >::firstControl()`.

**16.39.4.15** `template<int p, int q> virtual Eigen::VectorXd teb::TebController< p, q >::firstState ( ) const [inline], [virtual]`

#### Returns

Copy of the first state

Implements [teb::BaseController](#).

Definition at line 130 of file `teb_controller.h`.

References `teb::TebController< p, q >::_state_seq`.

**16.39.4.16** `template<int p, int q> const StateVector& teb::TebController< p, q >::firstStateRef ( ) const [inline]`

#### Returns

Reference to the first state (read-only)

Definition at line 135 of file `teb_controller.h`.

References `teb::TebController< p, q >::_state_seq`.

**16.39.4.17** `template<int p, int q> Eigen::VectorXd teb::TebController< p, q >::getAbsoluteTimeVec ( ) const [virtual]`

Calculate vector of absolute times  $t = [0, \Delta T, 2\Delta T, \dots, n\Delta T]$ .

#### Returns

Row vector containing time information [ $n \times 1$ ].

Implements [teb::BaseController](#).

Definition at line 780 of file `teb_controller.hpp`.

**16.39.4.18** `template<int p, int q> void teb::TebController< p, q >::getActiveVertices ( ) [protected]`

Iterate through the TEB sequence/trajectory and determine if a vertex ([StateVertex](#), [ControlVertex](#) or [TimeDiff](#)) is completely fixed.

In that case the corresponding vertex is not active and can be skipped during optimization (Jacobian calculation, Hessian, ...). All active vertices (pointers) are stored into `TebController::_active_vertices`.

While iterating all vertices, the hessian index is determined and stored inside the corresponding vertex (see `TebVertex::setOptVecIdx()` and `TebVertex::getOptVecIdx()`). For example:

- The final state (goal state) is fixed for many problems, in that case it does not appear in the optimization vector and therefore it is not part of the Hessian. But if at least a single component is unfixed, the complete vertex (`TebSample` in this case) becomes active. It's hessian or optimization vector index is zero. The number of free/unfixed variables is  $q$ .

- The second sample has no fixed components. It's hessian or optimization vector index is  $q$  since the last vertex starts at zero and has  $q$  free variables.
- The following vertex starts at  $2 * q + p$  and so on ...
- The [TimeDiff](#)  $\Delta T$  is stored at the end (and therefore it has the largest index if unfixed).

**Todo** Add case for different state and control input sequence lengths (e.g. for move blocking)

Definition at line 729 of file `teb_controller.hpp`.

```
16.39.4.19 template<int p, int q> virtual double teb::TebController< p, q >::getDt ( ) const [inline],
           [virtual]
```

Returns

$\Delta T$

Implements [teb::BaseController](#).

Definition at line 196 of file `teb_controller.h`.

References `teb::TebController< p, q >::_dt`, and `teb::TimeDiff::dt()`.

```
16.39.4.20 template<int p, int q> virtual unsigned int teb::TebController< p, q >::getM ( ) const [inline],
           [virtual]
```

Returns

number of control vectors.

Definition at line 206 of file `teb_controller.h`.

References `teb::TebController< p, q >::_ctrl_seq`.

Referenced by `teb::TebController< p, q >::returnControlInputSequence()`.

```
16.39.4.21 template<int p, int q> virtual unsigned int teb::TebController< p, q >::getN ( ) const [inline],
           [virtual]
```

Returns

number of state vectors.

Implements [teb::BaseController](#).

Definition at line 201 of file `teb_controller.h`.

References `teb::TebController< p, q >::_state_seq`.

```
16.39.4.22 template<int p, int q> Eigen::Matrix< double,-1, 2 > teb::TebController< p, q >::getSampleOptVecIdxVMat (
) const
```

Get info matrix that contains the Hessian index (first column) and the number of free variables (second column) for all states and control inputs (row-wise).

This function is used for debugging purposes, e.g. to test [getActiveVertices\(\)](#).

**Remarks**

The Hessian index correspond to the row/col index of the first component of the current [VertexType](#) ([StateVertex](#) and [ControlVertex](#)). With the number of free variables, the position of the current [VertexType](#) inside the "big" composed Hessian is determined completely.

**Returns**

a copy of the hessian-idx-info-matrix [n x 2]

Definition at line 536 of file `teb_controller.hpp`.

```
16.39.4.23  template<int p, int q> Eigen::MatrixXd teb::TebController< p, q >::getStateCtrlInfoMat ( ) const
           [virtual]
```

Convert TEB to a full (p+q by n) matrix.

The new matrix is composed as follows:

- size: [p+q x n] (double), n=[getN\(\)](#)
- rows 1:p -> states
- rows p+1:p+q -> control inputs
- columns 1:n -> discrete samples

**Returns**

new copy of the TEB matrix [p+q x n]

Implements [teb::BaseController](#).

Definition at line 479 of file `teb_controller.hpp`.

```
16.39.4.24  template<int p, int q> Eigen::Matrix< bool, p+q,-1 > teb::TebController< p, q >::getTEBFixedMap ( ) const
```

Show fixed states of the TEB as a (p+q by n) matrix for debugging purposes.

The matrix is composed as follows:

- size: [p+q x n] (double)
- rows 1:p -> states
- rows p+1:p+q -> control inputs
- columns 1:n -> samples

**Returns**

a copy of the fixed state TEB info matrix [p+q x n]

Definition at line 508 of file `teb_controller.hpp`.

16.39.4.25 `template<int p, int q> const HyperGraph& teb::TebController< p, q >::graph ( ) const` `[inline]`

The hyper-graph stores all active vertices and edges (objectives and constraints) and is passed to the dedicated solver class. The hyper-graph is created within [buildOptimizationGraph\(\)](#). Users can add custom edges by subclassing and overriding [customOptimizationGraph\(\)](#). All active vertices are collected within [getActiveVertices\(\)](#).

#### Returns

Read-only reference to the hyper-graph instance.

Definition at line 236 of file `teb_controller.h`.

References `teb::TebController< p, q >::_graph`.

16.39.4.26 `template<int p, int q> void teb::TebController< p, q >::initOptimization ( )`

Do everything what is necessary before calling the actual optimization [optimizeTEB\(\)](#).

This wrapper function calls [buildOptimizationGraph\(\)](#) to construct the hyper-graph. Afterwards it collects all active vertices using [getActiveVertices\(\)](#).

In addition a few undesired config settings are checked (only if compiled in Debug mode).

Definition at line 689 of file `teb_controller.hpp`.

References `teb::CENTRAL`, and `PRINT_DEBUG_COND_ONCE`.

16.39.4.27 `template<int p, int q> void teb::TebController< p, q >::initTrajectory ( const Eigen::Ref< const StateVector > & x0, const Eigen::Ref< const StateVector > & xf, unsigned int n )` `[virtual]`

This basic implementation linearly interpolates the trajectory between start  $\mathbf{x}_0$  and final state  $\mathbf{x}_f$ .

Control inputs  $\mathbf{u}_i$  are initialized to zero. This method also updates the number of discrete samples (state and control input vector pairs)  $n$ .

All existing states and control inputs of the sequences `TebController::_state_seq` and `TebController::_ctrl_seq` are deleted first.

#### Warning

Be careful, if you override this function in a subclass make sure to call [HyperGraph::notifyGraphModified\(\)](#) inside.

#### Parameters

$x0$	copy of the current start state vector [p x 1]
$xf$	current final state vector [p x 1]
$n$	number of discrete samples (state and control input vector pairs) between start and goal

**Todo** More efficient implementation that keeps previously allocated memory

Definition at line 22 of file `teb_controller.hpp`.

16.39.4.28 `template<int p, int q> void teb::TebController< p, q >::insertStateControlInputPair ( unsigned int sample_idx, const Eigen::Ref< const StateVector > & state, const Eigen::Ref< const ControlVector > & control, bool predict_control=true )`

Insert a new discrete StateVector ControlInput pair to `TebController::_state_seq` and `TebController::_ctrl_seq` at position `sample_idx`.

Existing elements between `idx+1` and `n` ([getN\(\)](#)) are shifted to the interval `idx+1` and `n+1`.

if `predict_u` is `true`, a new control input `u` at position `sample_idx` is predicted using the class method `predictControl()`.

#### Warning

Be careful, if you override this function in a subclass make sure to call `HyperGraph::notifyGraphModified()` inside.

#### Parameters

<i>sample_idx</i>	position at which the <a href="#">StateVertex</a> and <a href="#">ControlVertex</a> should be inserted
<i>state</i>	State vector [ <code>p</code> x 1] to be added to <code>_state_seq</code>
<i>control</i>	Control input vector [ <code>q</code> x 1] to be added <code>_ctrl_seq</code>
<i>predict_control</i>	if <code>true</code> , the control is not taken from <code>sample</code> , but it is calculated using the method <code>predictControl()</code> [default= <code>true</code> ];

Definition at line 400 of file `teb_controller.hpp`.

```
16.39.4.29  template<int p, int q> virtual Eigen::VectorXd teb::TebController< p, q >::lastState ( ) const  [inline],
           [virtual]
```

#### Returns

Copy of the last state

Implements [teb::BaseController](#).

Definition at line 140 of file `teb_controller.h`.

References `teb::TebController< p, q >::_state_seq`.

```
16.39.4.30  template<int p, int q> const StateVector& teb::TebController< p, q >::lastStateRef ( ) const  [inline]
```

#### Returns

Reference to the last state (read-only)

Definition at line 145 of file `teb_controller.h`.

References `teb::TebController< p, q >::_state_seq`.

```
16.39.4.31  template<int p, int q> void teb::TebController< p, q >::optimizeTEB ( )
```

Start the actual TEB optimization routine.

The outer-loop resizes the trajectory (refer to [resizeTrajectory\(\)](#) for infos), initializes the optimization problem (e.g. by creating the hyper-graph), and it calls the [BaseSolver::solve\(\)](#) routine of the selected solver. The number of outer-loop iterations can be set in [Config::Teb::teb\\_iter](#).

**Todo** The current implementation is really inefficient, since in each outer-loop-iteration all hyper-graph edges are deleted and afterwards reinstantiated. Create a dedicated update function in the future to hot-start from a previous hyper-graph.

#### See Also

[initOptimization\(\)](#), [resizeTrajectory\(\)](#), [BaseSolver::solve\(\)](#), [setSolver\(\)](#)

Definition at line 312 of file `teb_controller.hpp`.

Referenced by `teb::TebController< p, q >::step()`.

16.39.4.32 `template<int p, int q> void teb::TebController< p, q >::predictControl ( Eigen::Ref< ControlVector > ctrl_out, const Eigen::Ref< const StateVector > & x1, const Eigen::Ref< const StateVector > & x2 ) [inline]`

This method can be used to predict the plant input  $\mathbf{u}_i$  that transite drive the system from  $\mathbf{x}_i$  to  $\mathbf{x}_{i+1}$ .

The method can be used e.g. in the trajectory initialization phase, or inseration/deletion of new discrete samples.

The current implementation just copies the previous prediction (which is set to 0 inside the method [initTrajectory\(\)](#)). Problem specific subclasses can override this function and can use the inverse of the discrete system dynamics to derive a function for  $\mathbf{u}_i$  with respect  $\mathbf{x}_i$  and  $\mathbf{x}_{i+1}$ .

E.g. for linear scalar systems:

$$\dot{x} = Ax + bu \quad (16.1)$$

$$x_{k+1} = x_k + \Delta T Ax_k + \Delta T b u_k \quad (16.2)$$

$$u_k = \frac{x_{k+1} - x_k - \Delta T A}{\Delta T b} \quad (16.3)$$

#### Parameters

<i>ctrl_out</i>	[output] predicted control input vector $\mathbf{u}_i$ [ $q \times 1$ ]
<i>x1</i>	vector state $\mathbf{x}_i$ at discrete time $i$ [ $p \times 1$ ]
<i>x2</i>	vector state $\mathbf{x}_{i+1}$ at discrete time $i+1$ [ $p \times 1$ ]

Definition at line 462 of file `teb_controller.hpp`.

16.39.4.33 `template<int p, int q> void teb::TebController< p, q >::pushBackStateControlInputPair ( const Eigen::Ref< const StateVector > & state, const Eigen::Ref< const ControlVector > & control ) [inline]`

#### Parameters

<i>state</i>	State vector [ $p \times 1$ ] to be added to <code>_state_seq</code>
<i>control</i>	Control input vector [ $q \times 1$ ] to be added <code>_ctrl_seq</code>

Definition at line 253 of file `teb_controller.h`.

References `teb::TebController< p, q >::_ctrl_seq`, and `teb::TebController< p, q >::_state_seq`.

16.39.4.34 `template<int p, int q> void teb::TebController< p, q >::pushFrontStateControlInputPair ( const Eigen::Ref< const StateVector > & state, const Eigen::Ref< const ControlVector > & control ) [inline]`

#### Parameters

<i>state</i>	State vector [ $p \times 1$ ] to be added to <code>_state_seq</code>
<i>control</i>	Control input vector [ $q \times 1$ ] to be added <code>_ctrl_seq</code>

Definition at line 264 of file `teb_controller.h`.

References `teb::TebController< p, q >::_ctrl_seq`, and `teb::TebController< p, q >::_state_seq`.

16.39.4.35 `template<int p, int q> void teb::TebController< p, q >::removeStateControlInputPair ( unsigned int sample_idx, bool predict_control = true )`

Remove an existing discrete state control input pair from the object properties `TebController::_state_seq` and `TebController::_ctrl_seq` at position `sample_idx`.

Existing elements between `sample_idx+1` and `n` (`getN()`) are shifted to the interval `idx` and `n-1`.

If `predict_control` is `true` A new control input  $u$  at position `sample_idx` is predicted using the class method [predictControl\(\)](#).

**Warning**

Be careful, if you override this function in a subclass make sure to call [HyperGraph::notifyGraphModified\(\)](#) inside.

**Parameters**

<i>sample_idx</i>	index in <a href="#">TebController::_state_seq</a> and <a href="#">TebController::_ctrl_seq</a> to be deleted
<i>predict_control</i>	if <code>true</code> , the control for the new interval is calculated using the method <a href="#">predictControl()</a> [default= <code>true</code> ];

Definition at line 351 of file `teb_controller.hpp`.

**16.39.4.36** `template<int p, int q> void teb::TebController< p, q >::resampleTrajectory ( unsigned int n_new )`

This function changes the number of samples by resampling the complete trajectories [TebController::\\_state\\_seq](#) and [TebController::\\_ctrl\\_seq](#) (  $\mathbf{x}_i$ ,  $\mathbf{u}_i$ ,  $\Delta T$  ).

Currently only linear interpolation is implemented.

This function may called withing [resizeTrajectory\(\)](#) that is part of the TEB optimization loop (see [optimizeTEB\(\)](#)).

To add/remove a single state refer to [insertSample\(\)](#) / [removeSample\(\)](#).

**Warning**

Be careful, if you override this function in a subclass make sure to call [HyperGraph::notifyGraphModified\(\)](#) inside.

**Parameters**

<i>n_new</i>	new number of samples
--------------	-----------------------

**Todo** More efficient strategy without copying the complete sequences.

@ todo If someone freezes states within the sequence (in addition to start and goal), it is not set to unfreezed here... But we do not want to force all states to be unfreezed due to lack of efficiency...

**Todo** : later this should be `n_new-1`

Definition at line 120 of file `teb_controller.hpp`.

**16.39.4.37** `template<int p, int q> virtual void teb::TebController< p, q >::resetController ( ) [inline], [virtual]`

Implements [teb::BaseController](#).

Definition at line 357 of file `teb_controller.h`.

References [teb::TebController< p, q >::\\_ctrl\\_seq](#), [teb::TebController< p, q >::\\_dt](#), [teb::TebController< p, q >::\\_dt\\_ref](#), [teb::TebController< p, q >::\\_graph](#), [teb::TebController< p, q >::\\_state\\_seq](#), [teb::TebController< p, q >::cfg](#), [teb::HyperGraph::clearGraph\(\)](#), [teb::TimeDiff::dt\(\)](#), [teb::Config::Teb::dt\\_ref](#), and [teb::Config::teb](#).

**16.39.4.38** `template<int p, int q> void teb::TebController< p, q >::resizeTrajectory ( )`

Resizing the trajectory is helpful e.g.

for the following scenarios:

- Large disturbances or obstacles requires the teb to be extended in order to satisfy the given sample rate and avoid undesirable behavior due to a large/small discretization step width  $\Delta T$ . After clearance of disturbances/obstacles the teb should contract to its (time-)optimal trajectory again.
- If the distance to the goal state is getting smaller, dt is decreasing as well. This leads to a heavily fine-grained discretization in combination with many discrete samples. Thus, the computation time will be/remain high and in addition numerical instabilities can appear (e.g. due to the division by a small  $\Delta T$ ).

The implemented strategy observes the timediff  $\Delta T$ . and

- inserts a new sample if  $\Delta T > \Delta T_{ref} + \Delta T_{hyst}$
- removes a sample if  $\Delta T < \Delta T_{ref} - \Delta T_{hyst}$

$\Delta T_{ref}$  and  $\Delta T_{hyst}$  can be changed in [Config::teb](#).

Since the  $\Delta T$  is currently chosen uniformly, no information about a possible position (at which the new sample should be inserted) can be obtained. This function resamples linearly the whole trajectory using [resampleTrajectory\(\)](#).

If the time difference vertex [TimeDiff](#) is fixed during optimization ([TimeDiff::\\_fixed == true](#), [TimeDiff::isFixedAll\(\)](#)), this method returns without any changes.

Definition at line 86 of file `teb_controller.hpp`.

```
16.39.4.39  template<int p, int q> virtual Eigen::MatrixXd teb::TebController< p, q >::returnControlInputSequence ( )
           const [inline], [virtual]
```

#### Returns

Matrix containing all planned control inputs [q x n-1]

**Todo** Here we assume, that the last control input is invalid - We need to change this, if we have different sizes for state and control input sequences

Implements [teb::BaseController](#).

Definition at line 211 of file `teb_controller.h`.

References [teb::TebController< p, q >::\\_ctrl\\_seq](#), and [teb::TebController< p, q >::getM\(\)](#).

```
16.39.4.40  template<int p, int q> void teb::TebController< p, q >::setFixedDt ( bool fixed ) [inline]
```

#### See Also

[setupHorizon\(\)](#), [setGoalStatesFixedOrUnfixed\(\)](#)

#### Parameters

<i>fixed</i>	Set to <code>true</code> in order to fix <a href="#">TebController::_dt</a> .
--------------	---

Definition at line 583 of file `teb_controller.h`.

References [teb::TebController< p, q >::\\_dt](#), [teb::TebController< p, q >::\\_graph](#), [teb::HyperGraph::notifyGraphModified\(\)](#), and [teb::TimeDiff::setFixedAll\(\)](#).

Referenced by [teb::TebController< p, q >::setupHorizon\(\)](#).

```
16.39.4.41  template<int p, int q> void teb::TebController< p, q >::setFixedGoal ( bool fixed ) [inline],
           [protected]
```



## Parameters

<i>fixed</i>	Set to <code>true</code> in order to fix the states.
--------------	--

Definition at line 672 of file `teb_controller.h`.

References `teb::TebController< p, q >::_graph`, `teb::TebController< p, q >::_state_seq`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.42** `template<int p, int q> void teb::TebController< p, q >::setFixedGoal ( const bool * fixed ) [inline], [protected]`

## Parameters

<i>fixed</i>	Boolean array with <i>p</i> components. Set i-th component to <code>true</code> in order to fix the corresponding state.
--------------	--

Definition at line 684 of file `teb_controller.h`.

References `teb::TebController< p, q >::_graph`, `teb::TebController< p, q >::_state_seq`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.43** `template<int p, int q> void teb::TebController< p, q >::setFixedGoal ( const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> & fixed ) [inline], [protected]`

## Parameters

<i>fixed</i>	<code>Eigen::Vector</code> with <i>p</i> booleans. Set i-th component to <code>true</code> in order to fix the corresponding state.
--------------	---

Definition at line 696 of file `teb_controller.h`.

References `teb::TebController< p, q >::_graph`, `teb::TebController< p, q >::_state_seq`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.44** `template<int p, int q> void teb::TebController< p, q >::setFixedStart ( bool fixed ) [inline], [protected]`

## Parameters

<i>fixed</i>	Set to <code>true</code> in order to fix the states.
--------------	--

Definition at line 648 of file `teb_controller.h`.

References `teb::TebController< p, q >::_graph`, `teb::TebController< p, q >::_state_seq`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.45** `template<int p, int q> void teb::TebController< p, q >::setFixedStart ( const bool * fixed ) [inline], [protected]`

## Parameters

<i>fixed</i>	Boolean array with <i>p</i> components. Set i-th component to <code>true</code> in order to fix the corresponding state.
--------------	--

Definition at line 660 of file `teb_controller.h`.

References `teb::TebController< p, q >::_graph`, `teb::TebController< p, q >::_state_seq`, and `teb::HyperGraph::notifyGraphModified()`.

16.39.4.46 `template<int p, int q> void teb::TebController< p, q >::setGoalStatesFixedOrUnfixed ( const bool *  
fixed_goal_states ) [inline]`

## Parameters

<i>fixed_goal_states</i>	boolean array of size <code>p</code> . Each component of <code>lastState()</code> is set to <code>fixed()</code> if <code>fixed_goal_states[i] == true</code> .
--------------------------	---

Definition at line 273 of file `teb_controller.h`.

References `teb::TebController< p, q >::_fixed_goal_states`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

Referenced by `teb::TebController< p, q >::setupHorizon()`.

16.39.4.47 `template<int p, int q> void teb::TebController< p, q >::setGoalStatesFixedOrUnfixed ( const Eigen::Ref< const Eigen::Matrix< bool, p, 1 >> & fixed_goal_states ) [inline]`

## Parameters

<i>fixed_goal_states</i>	<code>Eigen::Vector</code> of size <code>p</code> containing booleans. Each component of <code>lastState()</code> is set to <code>fixed()</code> if <code>fixed_goal_states[i] == true</code> .
--------------------------	---

Definition at line 283 of file `teb_controller.h`.

References `teb::TebController< p, q >::_fixed_goal_states`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

16.39.4.48 `template<int p, int q> void teb::TebController< p, q >::setGoalStatesFixedOrUnfixed ( bool all_goal_states_fixed ) [inline]`

## See Also

[setupHorizon\(\)](#), [setFixedDt\(\)](#)

## Parameters

<i>all_goal_states_fixed</i>	if <code>true</code> , all components of <code>lastState()</code> are set to <code>fixed()</code>
------------------------------	---

Definition at line 294 of file `teb_controller.h`.

References `teb::TebController< p, q >::_fixed_goal_states`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

16.39.4.49 `template<int p, int q> void teb::TebController< p, q >::setSolver ( BaseSolver * solver ) [inline]`

Set subclasses of [BaseSolver](#) here. Solver classes are frequently expanded.

## Remarks

Subclasses of [BaseSolverLeastSquares](#) are least-squares solvers that are efficient, but originally not suited for hard constraints. In addition, the objective function must be a quadratic function. In order to allow similar functionalities, the cost/objective function is squared  $\min V^2(\mathcal{B})$  and equality/inequality constraints are approximated by quadratic soft-constraints [5].

## Parameters

<i>solver</i>	Pointer to subclass of <a href="#">BaseSolver</a> . <a href="#">TebController</a> does not free memory of the solver class.
---------------	---

## See Also

[BaseSolver](#) [BaseSolverLeastSquares](#)

Definition at line 626 of file `teb_controller.h`.

References `teb::TebController< p, q >::_solver`, `teb::TebController< p, q >::cfg`, and `teb::BaseSolver::setConfig()`.

Referenced by `teb::TebController< p, q >::TebController()`.

**16.39.4.50** `template<int p, int q> void teb::TebController< p, q >::setSystemDynamics ( SystemDynamics< p, q > * system, bool activate = true ) [inline]`

Add system dynamics equations to the optimization problem. System equations are defined as a equality constraints since they should be satisfied all the time.

See [SystemDynamics](#) class for more details.

## Parameters

<i>system</i>	pointer to the <a href="#">SystemDynamics</a> (sub-)class that stores the system dynamics equations.
<i>activate</i>	if set to <code>true</code> than the edge is activated.

## See Also

[buildOptimizationGraph\(\)](#), [activateObjectiveTimeOptimal\(\)](#), [activateStateBounds\(\)](#), [activateControlBounds\(\)](#)

Definition at line 604 of file `teb_controller.h`.

References `teb::TebController< p, q >::_active_system_dynamics`, `teb::TebController< p, q >::_graph`, and `teb::HyperGraph::notifyGraphModified()`.

**16.39.4.51** `template<int p, int q> void teb::TebController< p, q >::setupHorizon ( bool fixed_goal, bool fixed_resolution, int no_samples = -1, double dt = -1 ) [inline]`

Use this function to setup different common horizon settings at once instead of changing config parameters and class members individually.

Common horizons:

- Receding/moving horizon: unfixed goal and fixed resolution of samples (fixed dt)
- Fixed end-point: fixed goal and fixed resolution of samples (fixed dt)
- TEB default: fixed goal and unfixed resolution (unfixed dt, in order to minimize time)

If you choose an unfixed resolution (`fixed_resolution == false`) make sure that there are objectives considering the time difference  $\Delta T$ , otherwise the problem will probably be ill-posed.

If you want to fix or unfix only a subset of goal states, use the method [setGoalStatesFixedOrUnfixed\(\)](#) after calling `setupHorizon` or without calling `setupHorizon`.

## See Also

[setFixedDt\(\)](#), [setGoalStatesFixedOrUnfixed\(\)](#)

## Parameters

<i>fixed_goal</i>	Set to <code>true</code> in order to fix or unfix all goal state variables.
<i>fixed_resolution</i>	Set to <code>true</code> in order to fix the resolution (the time difference $\Delta T$ between consecutive samples).
<i>no_samples</i>	Set default/initial number of samples (it remains constant, if the resolution is fixed as well). If -1, <a href="#">Config::Teb::n_pre</a> is used.
<i>dt</i>	Set reference time difference for discretization (it remains constant, if the resolution is fixed as well). If -1, <a href="#">Config::Teb::dt_ref</a> is used.

Definition at line 327 of file `teb_controller.h`.

References `teb::TebController< p, q >::_dt`, `teb::TebController< p, q >::_dt_ref`, `teb::TebController< p, q >::_no_samples`, `teb::TebController< p, q >::cfg`, `teb::TebController< p, q >::dt()`, `teb::TimeDiff::dt()`, `teb::Config::Teb::dt_min`, `teb::Config::Teb::dt_ref`, `teb::Config::Teb::n_min`, `teb::Config::Teb::n_pre`, `teb::TebController< p, q >::setFixedDt()`, `teb::TebController< p, q >::setGoalStatesFixedOrUnfixed()`, and `teb::Config::teb`.

**16.39.4.52** `template<int p, int q> const StateSequence& teb::TebController< p, q >::stateSequence ( ) const`  
[inline]

Returns

Reference to the state sequence (read-only)

Definition at line 181 of file `teb_controller.h`.

References `teb::TebController< p, q >::_state_seq`.

**16.39.4.53** `template<int p, int q> virtual void teb::TebController< p, q >::step ( const double *const x0, const double *const xf, double * ctrl_out = nullptr )` [inline], [virtual]

Parameters

<i>x0</i>	double array containing start state <i>x0</i> with <i>p</i> components [ <i>p</i> x 1]
<i>xf</i>	double array containing final state <i>xf</i> with <i>p</i> components [ <i>p</i> x 1]
<i>ctrl_out</i>	[output] store control input ( <a href="#">firstControl()</a> ) to control the plant [ <i>q</i> x 1]

Implements [teb::BaseController](#).

Definition at line 398 of file `teb_controller.h`.

**16.39.4.54** `template<int p, int q> void teb::TebController< p, q >::step ( const Eigen::Ref< const StateVector > & x0, const Eigen::Ref< const StateVector > & xf, Eigen::Map< ControlVector > * ctrl_out = nullptr )`  
[inline]

Calls [updateStart\(\)](#), [updateGoal\(\)](#), [optimizeTEB\(\)](#) and [firstControl\(\)](#).

Parameters

<i>x0</i>	StateVector containing start state <i>x0</i> with <i>p</i> components [ <i>p</i> x 1]
<i>xf</i>	StateVector containing final state <i>xf</i> with <i>p</i> components [ <i>p</i> x 1]
<i>ctrl_out</i>	[output] store control input ( <a href="#">firstControl()</a> ) to ControlVector-Map [ <i>q</i> x 1]

Definition at line 412 of file `teb_controller.h`.

References `teb::TebController< p, q >::firstControl()`, `teb::TebController< p, q >::optimizeTEB()`, `teb::TebController< p, q >::updateGoal()`, and `teb::TebController< p, q >::updateStart()`.

**16.39.4.55** `template<int p, int q> void teb::TebController< p, q >::updateGoal ( const Eigen::Ref< const StateVector > & xf )` [virtual]

Update goal/final StateVector.

If `getN()==1` (that is if only start state exists after setting `updateStart()`) than `initTrajectory()` is called to initialize a trajectory between previously defiend start and `xf`.

If an existing goal (`lastState()`) is far away (`Config::Teb::goal_dist_force_reinit`), the trajectory is reinitialized using `initTrajectory()` as well.

If the goal is not completely fixed during optimization, only unfixed components of the final state are updated.

#### Warning

Be careful, if you override this function in a subclass make sure to call `HyperGraph::notifyGraphModified()` inside.

#### Parameters

<code>xf</code>	new final state vector [p x 1]
-----------------	--------------------------------

Definition at line 261 of file `teb_controller.hpp`.

Referenced by `teb::TebController< p, q >::step()`.

```
16.39.4.56  template<int p, int q> void teb::TebController< p, q >::updateStart ( const Eigen::Ref< const StateVector
           > & x0 ) [virtual]
```

Update first/start `StateVector` (start of the trajectory) with a new one (`x_start`).

This can be obtained from the most recent measurements. The `teb` vector is pruned if a state with `idx i>0` is found, that has a smaller distance (l2-norm) to the new `x0` in comparance to `firstState()`.

If the `TEB` vector is empty, the start state is added anyway (as single state trajectory).

#### Warning

Be careful, if you override this function in a subclass make sure to call `HyperGraph::notifyGraphModified()` inside.

#### Parameters

<code>x0</code>	new start state vector [p x 1]
-----------------	--------------------------------

#### See Also

[updateGoal\(\)](#), [initTrajectory\(\)](#)

Definition at line 203 of file `teb_controller.hpp`.

Referenced by `teb::TebController< p, q >::step()`.

## 16.39.5 Member Data Documentation

```
16.39.5.1  template<int p, int q> std::tuple<bool, ControlVector, ControlVector> teb::TebController< p, q
           >::active_control_bounds = std::make_tuple(false,ControlVector::Constant(INF),ControlVector::Constant(-INF))
           [protected]
```

- First type (`bool`): if `true`, `EdgeControlBounds` is activated.
- Second type (`ControlVector`): lower bound on each `ControlVector`  $u_{min}$  (component-wise).
- Third type (`ControlVector`): upper bound on each `ControlVector`  $u_{max}$  (component-wise).

See Also

[EdgeControlBounds](#), [buildOptimizationGraph\(\)](#)

Definition at line 770 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::activateControlBounds()`, and `teb::TebController< p, q >::firstControlSaturated()`.

**16.39.5.2** `template<int p, int q> std::tuple<bool, double, double, double> teb::TebController< p, q >::active_quadratic_form = std::make_tuple(false,1,1,1) [protected]`

- First type (`bool`): if `true`, [EdgeQuadraticForm](#) is activated (quadratic cost on states and control inputs).
- Second type (`Weight Q`): Weight for the state vectors (uniform weight here)
- Third type (`Weight R`): Weight for the control input vectors (uniform weight here)
- Fourth type (`Weight Qf`): Weight for the final state vector (uniform weight here)

See Also

[activateObjectiveQuadraticForm\(\)](#), [EdgeControlBounds](#), [buildOptimizationGraph\(\)](#)

Definition at line 752 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::activateObjectiveQuadraticForm()`.

**16.39.5.3** `template<int p, int q> std::tuple<bool, StateVector, StateVector> teb::TebController< p, q >::active_state_bounds = std::make_tuple(false, StateVector::Constant(-INF), StateVector::Constant(INF)) [protected]`

- First type (`bool`): if `true`, [EdgeStateBounds](#) is activated.
- Second type (`StateVector`): lower bound on each `StateVector`  $x_{min}$  (component-wise).
- Third type (`StateVector`): upper bound on each `StateVector`  $x_{max}$  (component-wise).

See Also

[EdgeStateBounds](#), [buildOptimizationGraph\(\)](#)

Definition at line 779 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::activateStateBounds()`.

**16.39.5.4** `template<int p, int q> std::pair<bool, SystemDynamics<p, q>*> teb::TebController< p, q >::active_system_dynamics = std::make_pair(false,nullptr) [protected]`

- First type (`bool`): if `true`, [EdgeSystemDynamics](#) is activated.
- Second type (`SystemDynamics`): pointer to [SystemDynamics](#) object.

See Also

[setSystemDynamics\(\)](#), [SystemDynamics](#), [EdgeSystemDynamics](#), [buildOptimizationGraph\(\)](#)

Definition at line 761 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::setSystemDynamics()`.

**16.39.5.5** `template<int p, int q> bool teb::TebController< p, q >::_active_time_optimal = false` [protected]

Definition at line 742 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::activateObjectiveTimeOptimal()`.

**16.39.5.6** `template<int p, int q> bool teb::TebController< p, q >::_cfg_owned = false` [protected]

Definition at line 737 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::TebController()`, and `teb::TebController< p, q >::~~TebController()`.

**16.39.5.7** `template<int p, int q> ControlSequence teb::TebController< p, q >::_ctrl_seq` [protected]

Definition at line 712 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::controlSequence()`, `teb::TebController< p, q >::firstControl()`, `teb::TebController< p, q >::firstControlRef()`, `teb::TebController< p, q >::getM()`, `teb::TebController< p, q >::pushBackStateControlInputPair()`, `teb::TebController< p, q >::pushFrontStateControlInputPair()`, `teb::TebController< p, q >::resetController()`, and `teb::TebController< p, q >::returnControlInputSequence()`.

**16.39.5.8** `template<int p, int q> TimeDiff teb::TebController< p, q >::_dt` [protected]

Stores  $\Delta T$ . Another interpretation: discretization step width. This implementation supports currently a uniformly distributed `dt` only.

Definition at line 720 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::dt()`, `teb::TebController< p, q >::getDt()`, `teb::TebController< p, q >::resetController()`, `teb::TebController< p, q >::setFixedDt()`, and `teb::TebController< p, q >::setupHorizon()`.

**16.39.5.9** `template<int p, int q> double teb::TebController< p, q >::_dt_ref = -1` [protected]

Definition at line 726 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::resetController()`, and `teb::TebController< p, q >::setupHorizon()`.

**16.39.5.10** `template<int p, int q> FixedStates teb::TebController< p, q >::_fixed_goal_states = FixedStates::Constant(true)` [protected]

Definition at line 733 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::setGoalStatesFixedOrUnfixed()`.

**16.39.5.11** `template<int p, int q> StateVector teb::TebController< p, q >::_goal_backup` [protected]

Definition at line 723 of file `teb_controller.h`.

**16.39.5.12** `template<int p, int q> HyperGraph teb::TebController< p, q >::_graph` [protected]

Definition at line 731 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::activateControlBounds()`, `teb::TebController< p, q >::activateObjectiveQuadraticForm()`, `teb::TebController< p, q >::activateObjectiveTimeOptimal()`, `teb::TebController< p, q >::activateStateBounds()`, `teb::TebController< p, q >::graph()`, `teb::TebController< p, q >::resetController()`, `teb::TebController< p, q >::setFixedDt()`, `teb::TebController< p, q >::setFixedGoal()`, `teb::TebController< p, q >::`



`>::setFixedStart()`, `teb::TebController< p, q >::setGoalStatesFixedOrUnfixed()`, `teb::TebController< p, q >::setSystemDynamics()`, and `teb::TebController< p, q >::~~TebController()`.

**16.39.5.13** `template<int p, int q> int teb::TebController< p, q >::_no_samples = -1` `[protected]`

Definition at line 725 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::setupHorizon()`.

**16.39.5.14** `template<int p, int q> bool teb::TebController< p, q >::_optimized` `[protected]`

Definition at line 729 of file `teb_controller.h`.

**16.39.5.15** `template<int p, int q> BaseSolver* teb::TebController< p, q >::_solver = nullptr` `[protected]`

Definition at line 735 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::setSolver()`.

**16.39.5.16** `template<int p, int q> StateSequence teb::TebController< p, q >::_state_seq` `[protected]`

Definition at line 711 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::firstState()`, `teb::TebController< p, q >::firstStateRef()`, `teb::TebController< p, q >::getN()`, `teb::TebController< p, q >::lastState()`, `teb::TebController< p, q >::lastStateRef()`, `teb::TebController< p, q >::pushBackStateControlInputPair()`, `teb::TebController< p, q >::pushFrontStateControlInputPair()`, `teb::TebController< p, q >::resetController()`, `teb::TebController< p, q >::setFixedGoal()`, `teb::TebController< p, q >::setFixedStart()`, and `teb::TebController< p, q >::stateSequence()`.

**16.39.5.17** `template<int p, int q> const Config* teb::TebController< p, q >::cfg`

If the [Config](#) class is provided via [TebController::TebController\(\)](#) than the memory management is not touched by this class. If no [Config](#) class is provided manually, memory management is owned by this class.

See Also

[TebController::TebController\(\)](#), [TebController::cfg\\_owned](#)

Definition at line 90 of file `teb_controller.h`.

Referenced by `teb::TebController< p, q >::resetController()`, `teb::TebController< p, q >::setSolver()`, `teb::TebController< p, q >::setupHorizon()`, and `teb::TebController< p, q >::~~TebController()`.

**16.39.5.18** `template<int p, int q> const unsigned int teb::TebController< p, q >::NoControls = q` `[static]`

Definition at line 60 of file `teb_controller.h`.

**16.39.5.19** `template<int p, int q> const unsigned int teb::TebController< p, q >::NoStates = p` `[static]`

Definition at line 59 of file `teb_controller.h`.

The documentation for this class was generated from the following files:

- [teb\\_controller.h](#)
- [teb\\_controller.hpp](#)

## 16.40 teb::TebPlotter Class Reference

Provides useful visualization and plotting functions.

```
#include <teb_plotter.h>
```

### Public Types

- enum [FileFormat](#) {  
[FileFormat::PNG](#), [FileFormat::JPEG](#), [FileFormat::SVG](#), [FileFormat::PDF](#),  
[FileFormat::EPS](#), [FileFormat::TIKZ](#) }  
*Supported file formats for figure export.*

### Public Member Functions

- [TebPlotter](#) ()  
*Constructor.*
- [~TebPlotter](#) ()  
*Destructor.*
- void [plot1DVector](#) (const Eigen::Ref< const Eigen::VectorXd > &data)  
*Plot a single vector over the range 0 .. n.*
- void [plot](#) (const Eigen::Ref< const Eigen::VectorXd > &x, const Eigen::Ref< const Eigen::VectorXd > &y, std::string title="", std::string xlabel="", std::string ylabel="")  
*Create a common 2D plot.*
- void [spyMatrix](#) (const Eigen::Ref< const Eigen::MatrixXd > &matrix)  
*Plot the sparsity pattern of a given matrix.*
- template<typename StateSeq, typename ControlSeq >  
void [plotTEB](#) (const StateSeq &state\_seq, ControlSeq &ctrl\_seq, double dt)  
*Plot TEB states and control inputs.*
- template<typename TebCtrl >  
void [plotTEB](#) (const TebCtrl &teb)  
*Plot TEB states and control inputs.*
- void [plotTwoCol](#) (const Eigen::Ref< const Eigen::VectorXd > &col1\_time, const Eigen::Ref< const Eigen::MatrixXd > &col1\_data, const Eigen::Ref< const Eigen::VectorXd > &col2\_time, const Eigen::Ref< const Eigen::MatrixXd > &col2\_data, [PlotOptions](#) \*options=nullptr)  
*Create a two column multiplot with a single line for each plot.*
- void [plotTwoCol](#) (const std::vector< const Eigen::VectorXd \* > &col1\_time, const std::vector< const Eigen::MatrixXd \* > &col1\_data, const std::vector< const Eigen::VectorXd \* > &col2\_time, const std::vector< const Eigen::MatrixXd \* > &col2\_data, [PlotOptions](#) \*options=nullptr)  
*Create a two column multiplot with multiple lines for each plot.*
- void [plotMulti](#) (const Eigen::Ref< const Eigen::VectorXd > &col1\_time, const Eigen::Ref< const Eigen::MatrixXd > &col1\_data, [PlotOptions](#) \*options=nullptr)  
*Create a one column multiplot with a single line for each plot.*
- void [plotMulti](#) (const std::vector< const Eigen::VectorXd \* > &col1\_time, const std::vector< const Eigen::MatrixXd \* > &col1\_data, [PlotOptions](#) \*options=nullptr)  
*Create a one column multiplot with multiple lines for each plot.*
- void [clearWindow](#) ()  
*Clear current gnuplot window.*
- void [switchWindow](#) (int window\_id, bool reset=true)  
*Switch gnuplot window or create a new one.*
- void [closeWindow](#) (int window\_id=0)  
*Close gnuplot window with id window\_id.*

- void `setOutputToFile` (std::string filename, `FileFormat` format=`FileFormat::PNG`)  
*Change output to file.*
- void `completePdfExport` ()  
*Use this function to complete the pdf export after plotting.*
- void `setOutputToWindow` ()  
*Change output to a common gnuplot window.*
- bool `isExportToFileEnabled` ()  
*Query status if export to file is enabled.*

### Protected Member Functions

- void `plotCustomKey` (const std::vector< std::string > &keys)  
*Plot a legend into a multiplot environment.*

### Protected Attributes

- bool `_pdf_flag` = false  
*true if PDF export is enabled.*
- bool `_file_export` = false  
*true if export to file is desired*
- FILE \* `pipe` = nullptr

#### 16.40.1 Detailed Description

This class provides visualization and plot functions for dynamic systems and the `TebController`. In addition this class defines plot functions for custom data specified by `Eigen::Vector` types.

Requirements: This class needs gnuplot installed correctly and it must be determined by cmake first (GNUPLOT\_PATH needs to be valid).

**Bug** EPS and PDF export are not working as expected. No text is displayed and after importing to Inksape, the plot is blank (noticed on Ubuntu 14.04).

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### Examples:

`integrator_system1.cpp`, `integrator_system2.cpp`, `integrator_system_classic_mpc.cpp`, `linear_system_ode.cpp`, `linear_system_ode_ctrl_comparison.cpp`, `linear_system_state_space.cpp`, `mobile_robot_teb.cpp`, `rocket_system.cpp`, and `van_der_pol_system.cpp`.

Definition at line 56 of file `teb_plotter.h`.

#### 16.40.2 Member Enumeration Documentation

16.40.2.1 enum `teb::TebPlotter::FileFormat` [strong]

#### Enumerator

**PNG**

**JPEG**

**SVG****PDF****EPS****TIKZ**

Definition at line 61 of file `teb_plotter.h`.

### 16.40.3 Constructor & Destructor Documentation

#### 16.40.3.1 `teb::TebPlotter::TebPlotter ( )`

Constructs a new [TebPlotter](#) object.

A new pipe to gnuplot is initialized. The default output is set to a new window.

Definition at line 13 of file `teb_plotter.cpp`.

References `pipe`, `PRINT_INFO`, and `setOutputToWindow()`.

#### 16.40.3.2 `teb::TebPlotter::~~TebPlotter ( )`

Destructs the [TebPlotter](#) object.

The pipe to gnuplot is closed.

Definition at line 38 of file `teb_plotter.cpp`.

References `completePdfExport()`, and `pipe`.

### 16.40.4 Member Function Documentation

#### 16.40.4.1 `void teb::TebPlotter::clearWindow ( )` `[inline]`

Definition at line 121 of file `teb_plotter.h`.

References `pipe`.

#### 16.40.4.2 `void teb::TebPlotter::closeWindow ( int window_id = 0 )` `[inline]`

##### Parameters

<code><i>window_id</i></code>	that should be closed.
-------------------------------	------------------------

Definition at line 146 of file `teb_plotter.h`.

References `pipe`.

#### 16.40.4.3 `void teb::TebPlotter::completePdfExport ( )` `[inline]`

Definition at line 216 of file `teb_plotter.h`.

References `_pdf_flag`, and `pipe`.

Referenced by `~TebPlotter()`.

#### 16.40.4.4 `bool teb::TebPlotter::isExportToFileEnabled ( )` `[inline]`

**Returns**

`true`, if export to file is enabled

Definition at line 243 of file `teb_plotter.h`.

References `_file_export`.

**16.40.4.5** `void teb::TebPlotter::plot ( const Eigen::Ref< const Eigen::VectorXd > & x, const Eigen::Ref< const Eigen::VectorXd > & y, std::string title = " ", std::string xlabel = " ", std::string ylabel = " " )`

Create a simple 2D plot using two different n-dim vectors for x and y.

**Parameters**

<i>x</i>	x-data vector (abscissa): <code>Eigen::Vector [n x 1]</code>
<i>y</i>	y-data vector (ordinate): <code>Eigen::Vector [n x 1]</code>
<i>title</i>	String defining the title of the plot (optional)
<i>xlabel</i>	String defining the label of the x-axis (optional)
<i>ylabel</i>	String defining the label of the y-axis (optional)

disable legend

show grid

plot title

plot xlabel

plot ylabel

plot type

loop over the data

data terminated with

termination character

flush the pipe

Definition at line 81 of file `teb_plotter.cpp`.

References `pipe`, and `PRINT_INFO`.

**16.40.4.6** `void teb::TebPlotter::plot1DVector ( const Eigen::Ref< const Eigen::VectorXd > & data )`

Plot a single vector.

The abscissa is chosen uniformly to 0 .. n

**Parameters**

<i>data</i>	Y data wrapped into an <code>Eigen::Vector [n x 1]</code>
-------------	---

disable legend

show grid

plot type

loop over the data

data terminated with

termination character

flush the pipe

Definition at line 55 of file `teb_plotter.cpp`.

References `pipe`, and `PRINT_INFO`.

16.40.4.7 `void teb::TebPlotter::plotCustomKey ( const std::vector< std::string > & keys )` [protected]

Use this function to plot a custom legend into a multicol environment.

\ Specify location before calling this function by setting gnuplot margins `tmargin`, `bmargin`, `lmargin` and `rmargin`.

#### Parameters

<i>keys</i>	Vector of legend strings (for each line)
-------------	--

WORKS ONLY IN MULTI PLOT ENVIRONMENT!

terminate

Definition at line 574 of file `teb_plotter.cpp`.

References `pipe`.

Referenced by `plotTwoCol()`.

16.40.4.8 `void teb::TebPlotter::plotMulti ( const Eigen::Ref< const Eigen::VectorXd > & col1_time, const Eigen::Ref< const Eigen::MatrixXd > & col1_data, PlotOptions * options = nullptr )` [inline]

Definition at line 103 of file `teb_plotter.h`.

References `plotTwoCol()`.

16.40.4.9 `void teb::TebPlotter::plotMulti ( const std::vector< const Eigen::VectorXd * > & col1_time, const std::vector< const Eigen::MatrixXd * > & col1_data, PlotOptions * options = nullptr )` [inline]

Definition at line 111 of file `teb_plotter.h`.

References `plotTwoCol()`.

16.40.4.10 `template<typename StateSeq, typename ControlSeq> void teb::TebPlotter::plotTEB ( const StateSeq & state_seq, ControlSeq & ctrl_seq, double dt )`

Use this function to plot the state and control input sequence of the [TebController](#) given as `TebController::TEBVector` type.

#### Remarks

Based on <http://stackoverflow.com/questions/14712251/place-key-below-multiplot-graph-in> and extended to two column multiplots

#### Parameters

<i>state_seq</i>	State sequence (Vectors of type <a href="#">StateVertex</a> )
<i>ctrl_seq</i>	Ccontrol input sequence (Vectors of type <a href="#">ControlVertex</a> )
<i>dt</i>	Time interval between two consecutive states and control inputs $\Delta T$ .

#### Template Parameters

<i>TebVec</i>	Type of the <code>TebVec</code> (deduced by the compiler).
---------------	--

#### Examples:

[integrator\\_system1.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_state\\_space.cpp](#), and [van\\_der\\_pol\\_system.-cpp](#).

Definition at line 19 of file `teb_plotter.hpp`.

References pipe, and PRINT\_INFO.

Referenced by plotTEB().

16.40.4.11 `template<typename TebCtrl > void teb::TebPlotter::plotTEB ( const TebCtrl & teb ) [inline]`

Parameters

<i>teb</i>	Reference to a <a href="#">TebController</a> object
------------	---

Template Parameters

<i>Type</i>	of the <a href="#">TebController</a> object (deduced by the compiler)
-------------	---

Definition at line 87 of file `teb_plotter.h`.

References plotTEB().

16.40.4.12 `void teb::TebPlotter::plotTwoCol ( const Eigen::Ref< const Eigen::VectorXd > & col1_time, const Eigen::Ref< const Eigen::MatrixXd > & col1_data, const Eigen::Ref< const Eigen::VectorXd > & col2_time, const Eigen::Ref< const Eigen::MatrixXd > & col2_data, PlotOptions * options = nullptr )`

The number of rows for each column may differ.

Remarks

No legend options implemented

**Test** What happens if the right column contains more rows than the left one? (Alignment)

Parameters

<i>col1_time</i>	time axis for the left column [n1 x 1]
<i>col1_data</i>	data for the left column: Each row corresponds to a single plot [rows1 x n1]
<i>col2_time</i>	time axis for the right column [n2 x 1]
<i>col2_data</i>	data for the right column: Each row corresponds to a single plot [rows2 x n2]
<i>options</i>	Pointer to customized plot options given by a <a href="#">PlotOptions</a> object.

Definition at line 122 of file `teb_plotter.cpp`.

References pipe, PRINT\_INFO, `teb::PlotOptions::skip_last_value_right_column`, `teb::PlotOptions::title`, and `teb::PlotOptions::ylabels`.

Referenced by plotMulti().

16.40.4.13 `void teb::TebPlotter::plotTwoCol ( const std::vector< const Eigen::VectorXd * > & col1_time, const std::vector< const Eigen::MatrixXd * > & col1_data, const std::vector< const Eigen::VectorXd * > & col2_time, const std::vector< const Eigen::MatrixXd * > & col2_data, PlotOptions * options = nullptr )`

The number of rows for each column may differ.

One element of the `col1` arguments stores the complete information for all subplots. Other elements extend each plot by new lines. E.g:

- `col1_data.at(0)` contains 3 rows, that results 3 subplots / rows for the left column. Each row specifies one line for each plot.
- `col1_data.at(1)` contains 3 rows as well. Each line is added to its corresponding subplots.  
This requires the same number of samples (columns) between `col1_data.at(i)` and `col1_time.at(i)`.

## Parameters

<i>col1_time</i>	container of time data for the lines in the left column
<i>col1_data</i>	container of y-data for the lines in the left column
<i>col2_time</i>	container of time data for the lines in the right column
<i>col2_data</i>	container of y-data for the lines in the right column
<i>options</i>	Pointer to customized plot options given by a <a href="#">PlotOptions</a> object.

Definition at line 276 of file `teb_plotter.cpp`.

References `teb::PlotOptions::legend`, `teb::PlotOptions::legend_entries`, `pipe`, `plotCustomKey()`, `PRINT_INFO`, `teb::PlotOptions::skip_last_value_right_column`, `teb::PlotOptions::title`, and `teb::PlotOptions::ylabels`.

**16.40.4.14** `void teb::TebPlotter::setOutputToFile ( std::string filename, FileFormat format = FileFormat::PNG )`  
`[inline]`

## Parameters

<i>filename</i>	Export figure to a file called <code>filename</code> into the current working directory or specify complete path.
<i>format</i>	Fileformat chosen from <a href="#">TebPlotter::FileFormat</a> enum.

## Examples:

[integrator\\_system\\_classic\\_mpc.cpp](#).

Definition at line 158 of file `teb_plotter.h`.

References `_file_export`, `_pdf_flag`, `EPS`, `JPEG`, `PDF`, `pipe`, `PRINT_DEBUG`, `PRINT_DEBUG_ONCE`, `SVG`, and `TIKZ`.

**16.40.4.15** `void teb::TebPlotter::setOutputToWindow ( )` `[inline]`

Definition at line 226 of file `teb_plotter.h`.

References `pipe`.

Referenced by `TebPlotter()`.

**16.40.4.16** `void teb::TebPlotter::spyMatrix ( const Eigen::Ref< const Eigen::MatrixXd > & matrix )`

This function visualizes the non-zeros of a given sparse matrix with a marker at the specific non-zero position.

## Parameters

<i>matrix</i>	Matrix given as an <code>Eigen::Matrix</code> data type.
---------------	--

Definition at line 529 of file `teb_plotter.cpp`.

References `pipe`, and `PRINT_INFO`.

**16.40.4.17** `void teb::TebPlotter::switchWindow ( int window_id, bool reset = true )` `[inline]`

## Parameters

<i>window_id</i>	Switch to gnuplot window with id <code>window_id</code> . If it does not exist, create new window.
------------------	--



<code>reset</code>	Reset window settings (recommand after switching from multiplots with extra sizes)
--------------------	--

Examples:

[linear\\_system\\_ode\\_ctrl\\_comparison.cpp](#).

Definition at line 133 of file `teb_plotter.h`.

References `_file_export`, and `pipe`.

### 16.40.5 Member Data Documentation

**16.40.5.1** `bool teb::TebPlotter::_file_export = false` `[protected]`

Definition at line 251 of file `teb_plotter.h`.

Referenced by `isExportToFileEnabled()`, `setOutputToFile()`, and `switchWindow()`.

**16.40.5.2** `bool teb::TebPlotter::_pdf_flag = false` `[protected]`

Definition at line 250 of file `teb_plotter.h`.

Referenced by `completePdfExport()`, and `setOutputToFile()`.

**16.40.5.3** `FILE* teb::TebPlotter::pipe = nullptr` `[protected]`

Definition at line 253 of file `teb_plotter.h`.

Referenced by `clearWindow()`, `closeWindow()`, `completePdfExport()`, `plot()`, `plot1DVector()`, `plotCustomKey()`, `plotTEB()`, `plotTwoCol()`, `setOutputToFile()`, `setOutputToWindow()`, `spyMatrix()`, `switchWindow()`, `TebPlotter()`, and `~TebPlotter()`.

The documentation for this class was generated from the following files:

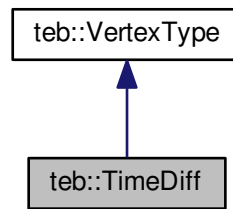
- [teb\\_plotter.h](#)
- [teb\\_plotter.hpp](#)
- [teb\\_plotter.cpp](#)

## 16.41 `teb::TimeDiff` Class Reference

Vertex that contains the time information of the TEB:  $\Delta T$ .

```
#include <teb_vertices.h>
```

Inheritance diagram for `teb::TimeDiff`:



## Public Types

- using `VertexContainer` = `std::vector< VertexType * >`  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*

## Public Member Functions

- `TimeDiff` (bool fixed=false)  
*Construct new `TimeDiff` and optionally set vertex fixed.*
- `TimeDiff` (double dt, bool fixed=false)  
*Construct new `TimeDiff` with specified initial value and optionally set vertex fixed.*
- `TimeDiff` (double \*dt, bool fixed=false)  
*Construct new `TimeDiff` with specified initial value and optionally set vertex fixed.*
- `TimeDiff` (const `TimeDiff` &obj)  
*Copy constructor.*
- `~TimeDiff` ()
- virtual bool `isFixedAll` () const  
*return true, if ALL values of this vertex are fixed and therefore are not necessary for optimization.*
- virtual void `setFixedAll` (bool fixed)  
*set `VertexType` fixed.*
- virtual bool `isFixedAny` () const  
*return true, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.*
- virtual bool `isFixedComp` (int) const  
*return true, if element/value/component with index idx of this vertex is fixed.*
- double & `dt` ()  
*return current  $\Delta T$ .*
- const double & `dt` () const  
*return current  $\Delta T$  (read-only).*
- virtual void `push` ()  
*This function should store all values into a internal backup stack.*
- virtual void `pop` ()  
*This function should restore the previously stored values of the backup stack and remove them from the stack.*
- virtual void `top` ()  
*This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.*

- virtual void `discardTop ()`  
*This function should delete the previously made backup from the stack without restoring it.*
- virtual unsigned int `stackSize () const`  
*This function should return the current size/number of backups of the backup stack.*
- virtual int `dimension () const`  
*Return number of elements/values/components stored in this vertex.*
- virtual int `dimensionFree () const`  
*Return only the dimension of free (unfixed) variables.*
- virtual void `plus (const VertexType *rhs)`  
*Define the increment for the vertex:  $x = x + rhs$ .*
- virtual void `plus (const double *rhs)`  
*Define the increment for the vertex:  $x = x + rhs$  ( $rhs[]$  with `dimension()`=p+q values)*
- virtual void `plusFree (const double *rhs)`  
*Define the increment for the vertex:  $x = x + rhs$  (But only add FREE variables,  $rhs[]$  with `dimensionFree()` values).*
- virtual void `setFree (const double *rhs)`  
*Overwrite all free variables with the values given by rhs ( $rhs[]$  with `dimensionFree()` values).*
- virtual void `getDataFree (double *target_vec) const`  
*Return a copy of the free (unfixed) values as a single array ( length: `dimensionFree()` ).*
- virtual const double & `getData (unsigned int) const`  
*Return pointer to data with the index `idx` ).*
- `TimeDiff & operator= (const TimeDiff &rhs)`
- void `operator() (double &dt)`
- void `setOptVecIdx (int opt_vec_idx)`  
*Update the position of the first value of this vertex inside the optimization vector / Hessian.*
- int `getOptVecIdx () const`  
*Get the position of the first value of this Vertex inside the optimization vector / Hessian.*

## Static Public Attributes

- static const int `Dimension = 1`  
*The dimension of the `TimeDiff` vertex is always 1.*

## Protected Attributes

- double `_dt = 0.1`
- bool `_fixed = false`
- std::stack< double > `_backup`
- int `_opt_vec_idx = -1`  
*Start-index in optimization vector (`idx` in hessian (row/column) and jacobian (column))*

## Friends

- std::ostream & `operator<< (std::ostream &os, const TimeDiff &dt)`  
*Print  $\Delta T$  using `std::ostream`.*

### 16.41.1 Detailed Description

#### Author

Christoph Rösman (christoph.roesmann@tu-dortmund.de)

#### See Also

[VertexType](#), [StateVertex](#), [ControlVertex](#), [BaseEdge](#)

Definition at line 518 of file teb\_vertices.h.

### 16.41.2 Member Typedef Documentation

16.41.2.1 using `teb::VertexType::VertexContainer = std::vector<VertexType*>` [inherited]

Definition at line 33 of file graph.h.

### 16.41.3 Constructor & Destructor Documentation

16.41.3.1 `teb::TimeDiff::TimeDiff ( bool fixed = false )` [inline]

#### Parameters

<i>fixed</i>	if <code>true</code> , the <a href="#">TimeDiff</a> is fixed during optimization (no time-optimal control possible).
--------------	--

Definition at line 527 of file teb\_vertices.h.

References `_fixed`.

16.41.3.2 `teb::TimeDiff::TimeDiff ( double dt, bool fixed = false )` [inline]

#### Parameters

<i>dt</i>	initial $\Delta T$
<i>fixed</i>	if <code>true</code> , the <a href="#">TimeDiff</a> is fixed during optimization (no time-optimal control possible).

Definition at line 534 of file teb\_vertices.h.

16.41.3.3 `teb::TimeDiff::TimeDiff ( double * dt, bool fixed = false )` [inline]

#### Parameters

<i>dt</i>	Pointer to initial $\Delta T$ (will be copied)
<i>fixed</i>	if <code>true</code> , the <a href="#">TimeDiff</a> is fixed during optimization (no time-optimal control possible).

Definition at line 541 of file teb\_vertices.h.

16.41.3.4 `teb::TimeDiff::TimeDiff ( const TimeDiff & obj )` [inline]

#### Parameters

<i>obj</i>	<a href="#">TimeDiff</a> object
------------	---------------------------------

Definition at line 547 of file `teb_vertices.h`.

**16.41.3.5** `teb::TimeDiff::~~TimeDiff ( )` `[inline]`

Definition at line 549 of file `teb_vertices.h`.

#### 16.41.4 Member Function Documentation

**16.41.4.1** `virtual int teb::TimeDiff::dimension ( ) const` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 574 of file `teb_vertices.h`.

**16.41.4.2** `virtual int teb::TimeDiff::dimensionFree ( ) const` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 575 of file `teb_vertices.h`.

References `_fixed`.

**16.41.4.3** `virtual void teb::TimeDiff::discardTop ( )` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 571 of file `teb_vertices.h`.

References `_backup`.

**16.41.4.4** `double& teb::TimeDiff::dt ( )` `[inline]`

Definition at line 556 of file `teb_vertices.h`.

References `_dt`.

Referenced by `teb::EdgeMinimizeTime::computeValues()`, `teb::TebController< p, q >::getDt()`, `operator()()`, `operator=()`, `plus()`, `teb::TebController< p, q >::resetController()`, and `teb::TebController< p, q >::setupHorizon()`.

**16.41.4.5** `const double& teb::TimeDiff::dt ( ) const` `[inline]`

Definition at line 557 of file `teb_vertices.h`.

References `_dt`.

**16.41.4.6** `virtual const double& teb::TimeDiff::getData ( unsigned idx ) const` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 609 of file `teb_vertices.h`.

References `_dt`.

16.41.4.7 `virtual void teb::TimeDiff::getDataFree ( double * target_vec ) const` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 600 of file `teb_vertices.h`.

References `_dt`, and `_fixed`.

16.41.4.8 `int teb::VertexType::getOptVecIdx ( ) const` `[inline], [inherited]`

Returns

position of the first element of this vertex.

Definition at line 59 of file `graph.h`.

References `teb::VertexType::_opt_vec_idx`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, and `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`.

16.41.4.9 `virtual bool teb::TimeDiff::isFixedAll ( ) const` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 551 of file `teb_vertices.h`.

References `_fixed`.

Referenced by operator=().

16.41.4.10 `virtual bool teb::TimeDiff::isFixedAny ( ) const` `[inline], [virtual]`

Implements [teb::VertexType](#).

Definition at line 553 of file `teb_vertices.h`.

References `_fixed`.

16.41.4.11 `virtual bool teb::TimeDiff::isFixedComp ( int idx ) const` `[inline], [virtual]`

Parameters

<i>idx</i>	component index
------------	-----------------

Implements [teb::VertexType](#).

Definition at line 554 of file `teb_vertices.h`.

References `_fixed`.

16.41.4.12 `void teb::TimeDiff::operator() ( double & dt )` `[inline]`

Definition at line 625 of file `teb_vertices.h`.

References `_dt`, and `dt()`.

16.41.4.13 **TimeDiff**& teb::TimeDiff::operator= ( const **TimeDiff** & *rhs* ) [inline]

Definition at line 615 of file teb\_vertices.h.

References `_dt`, `_fixed`, `dt()`, and `isFixedAll()`.

16.41.4.14 virtual void teb::TimeDiff::plus ( const **VertexType** \* *rhs* ) [inline],[virtual]

Implements [teb::VertexType](#).

Definition at line 578 of file teb\_vertices.h.

References `_dt`, and `dt()`.

16.41.4.15 virtual void teb::TimeDiff::plus ( const double \* *rhs* ) [inline],[virtual]

Implements [teb::VertexType](#).

Definition at line 584 of file teb\_vertices.h.

References `_dt`.

16.41.4.16 virtual void teb::TimeDiff::plusFree ( const double \* *rhs* ) [inline],[virtual]

Implements [teb::VertexType](#).

Definition at line 589 of file teb\_vertices.h.

References `_dt`, and `_fixed`.

16.41.4.17 virtual void teb::TimeDiff::pop ( ) [inline],[virtual]

Implements [teb::VertexType](#).

Definition at line 561 of file teb\_vertices.h.

References `_backup`, and `top()`.

16.41.4.18 virtual void teb::TimeDiff::push ( ) [inline],[virtual]

Implements [teb::VertexType](#).

Definition at line 560 of file teb\_vertices.h.

References `_backup`, and `_dt`.

16.41.4.19 virtual void teb::TimeDiff::setFixedAll ( bool *fixed* ) [inline],[virtual]

#### Parameters

<i>fixed</i>	if true <a href="#">TimeDiff</a> is fixed.
--------------	--

Definition at line 552 of file teb\_vertices.h.

References `_fixed`.

Referenced by `teb::TebController< p, q >::setFixedDt()`.

16.41.4.20 `virtual void teb::TimeDiff::setFree ( const double * rhs ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 594 of file `teb_vertices.h`.

References `_dt`, and `_fixed`.

16.41.4.21 `void teb::VertexType::setOptVecIdx ( int opt_vec_idx ) [inline],[inherited]`

#### Parameters

<code>opt_vec_idx</code>	index of the first value
--------------------------	--------------------------

Definition at line 53 of file `graph.h`.

References `teb::VertexType::opt_vec_idx`.

16.41.4.22 `virtual unsigned int teb::TimeDiff::stackSize ( ) const [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 572 of file `teb_vertices.h`.

References `_backup`.

16.41.4.23 `virtual void teb::TimeDiff::top ( ) [inline],[virtual]`

Implements [teb::VertexType](#).

Definition at line 566 of file `teb_vertices.h`.

References `_backup`, and `_dt`.

Referenced by `pop()`.

## 16.41.5 Friends And Related Function Documentation

16.41.5.1 `std::ostream& operator<< ( std::ostream & os, const TimeDiff & dt ) [friend]`

#### Parameters

<code>os</code>	output stream object
<code>dt</code>	specific <a href="#">TimeDiff</a> object

#### Returns

output stream object including a formatted string containing state and control vector.

Definition at line 637 of file `teb_vertices.h`.

## 16.41.6 Member Data Documentation

16.41.6.1 `std::stack<double> teb::TimeDiff::_backup [protected]`

Definition at line 646 of file `teb_vertices.h`.

Referenced by `discardTop()`, `pop()`, `push()`, `stackSize()`, and `top()`.



16.41.6.2 `double teb::TimeDiff::_dt = 0.1` `[protected]`

Definition at line 644 of file `teb_vertices.h`.

Referenced by `dt()`, `getData()`, `getDataFree()`, `operator()()`, `operator=()`, `plus()`, `plusFree()`, `push()`, `setFree()`, and `top()`.

16.41.6.3 `bool teb::TimeDiff::_fixed = false` `[protected]`

Definition at line 645 of file `teb_vertices.h`.

Referenced by `dimensionFree()`, `getDataFree()`, `isFixedAll()`, `isFixedAny()`, `isFixedComp()`, `operator=()`, `plusFree()`, `setFixedAll()`, `setFree()`, and `TimeDiff()`.

16.41.6.4 `int teb::VertexType::_opt_vec_idx = -1` `[protected]`, `[inherited]`

Definition at line 69 of file `graph.h`.

Referenced by `teb::VertexType::getOptVecIdx()`, and `teb::VertexType::setOptVecIdx()`.

16.41.6.5 `const int teb::TimeDiff::Dimension = 1` `[static]`

Definition at line 521 of file `teb_vertices.h`.

The documentation for this class was generated from the following file:

- [teb\\_vertices.h](#)

## 16.42 teb::SimResults::TimeSeries Struct Reference

Store measurements of dynamic systems (states and control inputs) w.r.t.

```
#include <simulator.h>
```

### Public Attributes

- `Eigen::MatrixXd` [states](#)  
*Contains state vectors  $[p \times n]$ .*
- `Eigen::MatrixXd` [controls](#)  
*Contains control input vectors  $[q \times n]$ .*
- `Eigen::VectorXd` [time](#)  
*Time vector  $[0 \dots T]$   $[n \times 1]$ .*
- `double` [dt](#)  
*Store time difference  $\Delta T$  of the current TEB.*

### 16.42.1 Detailed Description

time The time is discretized into  $n$  discrete time samples

Definition at line 37 of file `simulator.h`.

### 16.42.2 Member Data Documentation

#### 16.42.2.1 `Eigen::MatrixXd teb::SimResults::TimeSeries::controls`

Definition at line 40 of file `simulator.h`.

Referenced by `teb::SimResults::allocateMemory()`, and `teb::SimResults::conservativeResize()`.

#### 16.42.2.2 `double teb::SimResults::TimeSeries::dt`

Definition at line 42 of file `simulator.h`.

#### 16.42.2.3 `Eigen::MatrixXd teb::SimResults::TimeSeries::states`

Definition at line 39 of file `simulator.h`.

Referenced by `teb::SimResults::allocateMemory()`, and `teb::SimResults::conservativeResize()`.

#### 16.42.2.4 `Eigen::VectorXd teb::SimResults::TimeSeries::time`

Definition at line 41 of file `simulator.h`.

Referenced by `teb::SimResults::allocateMemory()`, and `teb::SimResults::conservativeResize()`.

The documentation for this struct was generated from the following file:

- [simulator.h](#)

## 16.43 `teb::Config::Utilities` Struct Reference

Configurations for helper miscellaneous and utilities.

```
#include <config.h>
```

### 16.43.1 Detailed Description

Definition at line 138 of file `config.h`.

The documentation for this struct was generated from the following file:

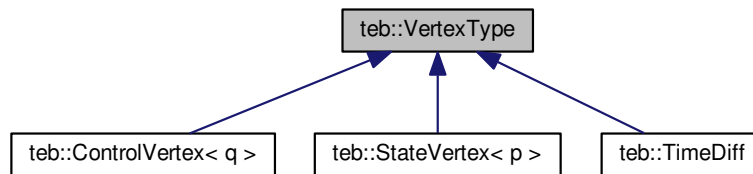
- [config.h](#)

## 16.44 `teb::VertexType` Class Reference

Generic interface class for vertices.

```
#include <graph.h>
```

Inheritance diagram for teb::VertexType:



## Public Types

- using [VertexContainer](#) = std::vector< [VertexType](#) \* >  
*Typedef for vertex containers that stores all vertices required for the optimization (Hyper-Graph).*

## Public Member Functions

- virtual bool [isFixedAll](#) () const =0  
*return true, if ALL values of this vertex are fixed and therefore are not necessary for optimization.*
- virtual bool [isFixedAny](#) () const =0  
*return true, if ANY element/value/component of this vertex is fixed and therefore the vertex is relevant for optimization.*
- virtual bool [isFixedComp](#) (int idx) const =0  
*return true, if element/value/component with index idx of this vertex is fixed.*
- virtual void [push](#) ()=0  
*This function should store all values into a internal backup stack.*
- virtual void [pop](#) ()=0  
*This function should restore the previously stored values of the backup stack and remove them from the stack.*
- virtual void [top](#) ()=0  
*This function should restore the previously stored values of the backup stack WITHOUT removing them from the stack.*
- virtual void [discardTop](#) ()=0  
*This function should delete the previously made backup from the stack without restoring it.*
- virtual unsigned int [stackSize](#) () const =0  
*This function should return the current size/number of backups of the backup stack.*
- virtual int [dimension](#) () const =0  
*Return number of elements/values/components stored in this vertex.*
- virtual int [dimensionFree](#) () const =0  
*Return only the dimension of free (unfixed) variables.*
- void [setOptVecIdx](#) (int opt\_vec\_idx)  
*Update the position of the first value of this vertex inside the optimization vector / Hessian.*
- int [getOptVecIdx](#) () const  
*Get the position of the first value of this Vertex inside the optimization vector / Hessian.*
- virtual void [plus](#) (const [VertexType](#) \*rhs)=0  
*Define the increment for the vertex:  $x = x + rhs$ .*
- virtual void [plus](#) (const double \*rhs)=0  
*Define the increment for the vertex:  $x = x + rhs$  (rhs[] with [dimension\(\)](#)=p+q values)*

- virtual void [plusFree](#) (const double \*rhs)=0  
*Define the increment for the vertex:  $x = x + rhs$  (But only add FREE variables, rhs[] with [dimensionFree\(\)](#) values).*
- virtual void [setFree](#) (const double \*rhs)=0  
*Overwrite all free variables with the values given by rhs (rhs[] with [dimensionFree\(\)](#) values).*
- virtual void [getDataFree](#) (double \*target\_vec) const =0  
*Return a copy of the free (unfixed) values as a single array ( length: [dimensionFree\(\)](#) ).*
- virtual const double & [getData](#) (unsigned int idx) const =0  
*Return pointer to data with the index `idx` ).*

## Protected Attributes

- int [\\_opt\\_vec\\_idx](#) = -1  
*Start-index in optimization vector (idx in hessian (row/column) and jacobian (column))*

### 16.44.1 Detailed Description

This abstract class defines the interface for dedicated vertices. The underlying TEB optimization problem is formulated as a hyper-graph in which state vectors ([StateVertex](#)), control input vectors ([ControlVertex](#)) and a [TimeDiff](#) are represented as vertices. That means, vertices have to be considered by the solver as changeable quantities.

#### Author

Christoph Rösman ( [christoph.roesmann@tu-dortmund.de](mailto:christoph.roesmann@tu-dortmund.de) )

#### See Also

[HyperGraph](#), [EdgeType](#), [StateVertex](#), [ControlVertex](#), [TimeDiff](#), [BaseEdge](#)

Definition at line 29 of file graph.h.

### 16.44.2 Member Typedef Documentation

16.44.2.1 using `teb::VertexType::VertexContainer = std::vector<VertexType*>`

Definition at line 33 of file graph.h.

### 16.44.3 Member Function Documentation

16.44.3.1 virtual int `teb::VertexType::dimension ( ) const` [pure virtual]

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, `teb::BaseEdge< D, FuncType, Vertices >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::BaseEdge< D, FuncType, Vertices >::computeJacobian()`, `teb::BaseEdge< D, FuncType >::computeJacobian()`, and `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`.

16.44.3.2 virtual int `teb::VertexType::dimensionFree ( ) const` [pure virtual]

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

16.44.3.3 `virtual void teb::VertexType::discardTop ( ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

16.44.3.4 `virtual const double& teb::VertexType::getData ( unsigned int idx ) const [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

16.44.3.5 `virtual void teb::VertexType::getDataFree ( double * target_vec ) const [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

16.44.3.6 `int teb::VertexType::getOptVecIdx ( ) const [inline]`

#### Returns

position of the first element of this vertex.

Definition at line 59 of file `graph.h`.

References `_opt_vec_idx`.

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, and `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`.

16.44.3.7 `virtual bool teb::VertexType::isFixedAll ( ) const [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseEdge< D, FuncType, Vertices >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::BaseEdge< D, FuncType, Vertices >::computeJacobian()`, and `teb::BaseEdge< D, FuncType >::computeJacobian()`.

16.44.3.8 `virtual bool teb::VertexType::isFixedAny ( ) const [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildObjectiveGradient()`, and `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`.

16.44.3.9 `virtual bool teb::VertexType::isFixedComp ( int idx ) const [pure virtual]`

#### Parameters

<i>idx</i>	component index
------------	-----------------

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseSolverNonlinearProgramDense::buildEqualityConstraintJacobian()`, `teb::BaseSolverNonlinearProgramDense::buildInequalityConstraintJacobian()`, `teb::SolverLevenbergMarquardtEigenDense::buildJacobian()`, `teb::SolverLevenbergMarquardtEigenSparse::buildJacobian()`, `teb::BaseSolverNonlinear-`

ProgramDense::buildObjectiveGradient(), `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianNumerically()`, and `teb::SolverLevenbergMarquardtEigenSparse::countJacobianColNNZ()`.

**16.44.3.10** `virtual void teb::VertexType::plus ( const VertexType * rhs ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseEdge< D, FuncType, Vertices >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::BaseEdge< D, FuncType, Vertices >::computeJacobian()`, and `teb::BaseEdge< D, FuncType >::computeJacobian()`.

**16.44.3.11** `virtual void teb::VertexType::plus ( const double * rhs ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

**16.44.3.12** `virtual void teb::VertexType::plusFree ( const double * rhs ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

**16.44.3.13** `virtual void teb::VertexType::pop ( ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseEdge< D, FuncType, Vertices >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::BaseEdge< D, FuncType, Vertices >::computeJacobian()`, and `teb::BaseEdge< D, FuncType >::computeJacobian()`.

**16.44.3.14** `virtual void teb::VertexType::push ( ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

Referenced by `teb::BaseEdge< D, FuncType, Vertices >::computeHessian()`, `teb::BaseEdge< D, FuncType >::computeHessian()`, `teb::BaseEdge< D, FuncType, Vertices >::computeJacobian()`, and `teb::BaseEdge< D, FuncType >::computeJacobian()`.

**16.44.3.15** `virtual void teb::VertexType::setFree ( const double * rhs ) [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

**16.44.3.16** `void teb::VertexType::setOptVecIdx ( int opt_vec_idx ) [inline]`

#### Parameters

<i>opt_vec_idx</i>	index of the first value
--------------------	--------------------------

Definition at line 53 of file `graph.h`.

References `_opt_vec_idx`.

**16.44.3.17** `virtual unsigned int teb::VertexType::stackSize ( ) const [pure virtual]`

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

16.44.3.18 `virtual void teb::VertexType::top ( )` [pure virtual]

Implemented in [teb::TimeDiff](#), [teb::ControlVertex< q >](#), and [teb::StateVertex< p >](#).

#### 16.44.4 Member Data Documentation

16.44.4.1 `int teb::VertexType::_opt_vec_idx = -1` [protected]

Definition at line 69 of file `graph.h`.

Referenced by `getOptVecIdx()`, and `setOptVecIdx()`.

The documentation for this class was generated from the following file:

- [graph.h](#)





# Chapter 17

## File Documentation

### 17.1 base\_controller.h File Reference

```
#include <teb_package/base/config.h>
#include <memory>
#include <cmath>
#include <deque>
```

#### Classes

- class [teb::BaseController](#)  
*Base controller class (General Controller API)*

#### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

### 17.2 base\_edge.h File Reference

```
#include <teb_package/base/graph.h>
#include <teb_package/base/teb_vertices.h>
#include <teb_package/utilities/misc.h>
#include <array>
#include <teb_package/base/base_edge.hpp>
```

#### Classes

- class [teb::BaseEdge< D, FuncType, Vertices >](#)  
*Templated base edge class that stores an arbitrary number of values.*
- class [teb::BaseEdge< D, FuncType >](#)  
*Templated base edge class that stores an arbitrary number of values (Partial template specialization that allows changing vertices dimensions at runtime).*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.3 `base_edge.hpp` File Reference

```
#include <teb_package/base/base_edge.h>
```

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.4 `base_edge_dynamics.h` File Reference

```
#include <teb_package/base/base_edge.h>
#include <teb_package/base/system_dynamics.h>
```

## Classes

- class [teb::EdgeSystemDynamics< p, q, central >](#)

*Equality constraint edge for satisfying continious system dynamics specified by an [SystemDynamics](#) object.*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.5 `base_solver.h` File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/config.h>
#include <teb_package/base/graph.h>
#include <memory>
#include <cmath>
#include <deque>
#include <Eigen/Core>
#include <Eigen/StdVector>
```

## Classes

- class [teb::BaseSolver](#)

*Base class for solver implementations.*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.6 base\_solver\_least\_squares.cpp File Reference

```
#include <teb_package/base/base_solver_least_squares.h>
```

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.7 base\_solver\_least\_squares.h File Reference

```
#include <teb_package/base/base_solver.h>
#include <Eigen/Dense>
```

## Classes

- class [teb::BaseSolverLeastSquares](#)

*Extended base solver class for least squares optimizations.*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.8 base\_solver\_nonlinear\_program\_dense.cpp File Reference

```
#include <teb_package/solver/base_solver_nonlinear_program_dense.h>
```

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.9 base\_solver\_nonlinear\_program\_dense.h File Reference

```
#include <teb_package/base/base_solver.h>
#include <Eigen/Dense>
```

## Classes

- class [teb::BaseSolverNonlinearProgramDense](#)  
*Extended base solver class for nonlinear programs (dense version).*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.10 bound\_constraints.h File Reference

```
#include <teb_package/base/base_edge.h>
#include <type_traits>
#include <tuple>
```

## Classes

- struct [teb::CustomBoundData](#)  
*Datastruct for custom bound data that can be queried from the [BoundConstraint](#) class.*
- class [teb::BoundConstraint](#)< [Bound\\_type](#), [Vertex](#), [Bound\\_vars](#), [Index](#) >  
*This class captures general bound constraints on optimization variables / vertices.*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## Enumerations

- enum [teb::BOUND\\_TYPE](#) { [teb::BOUND\\_TYPE::LOWER](#), [teb::BOUND\\_TYPE::UPPER](#), [teb::BOUND\\_TYPE::LOWERUPPER](#) }  
*Enumerator class that stores all types of bounds for consideration in [BoundConstraint](#) class.*
- enum [teb::BOUND\\_VARS](#) { [teb::BOUND\\_VARS::ALL](#), [teb::BOUND\\_VARS::SINGLE](#) }  
*Enumerator class that stores all types of variables that can be bounded using [BoundConstraint](#) class.*

## 17.11 common\_teb\_edges.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/bound_constraints.h>
#include <teb_package/base/base_edge_dynamics.h>
#include <teb_package/base/system_dynamics.h>
```

## Classes

- class [teb::EdgeMinimizeTime](#)  
*Objective edge: minimize  $\Delta T$ .*
- class [teb::EdgeQuadraticForm< p, q >](#)  
*Objective edge: quadratic form for states and control input.*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## Typedefs

- using [teb::EdgePositiveTime](#) = BoundConstraint< BOUND\_TYPE::LOWER, TimeDiff, BOUND\_VARS::SINGLE, 0 >  
*Bound constraint edge for the time difference:  $\Delta T \geq \varepsilon$ .*
- template<int q>  
using [teb::EdgeControlBounds](#) = BoundConstraint< BOUND\_TYPE::LOWERUPPER, ControlVertex< q >, BOUND\_VARS::ALL >  
*Bound constraint edge for control inputs:  $\mathbf{u}_{min} \leq \mathbf{u}_k \leq \mathbf{u}_{max}$ .*
- template<int p>  
using [teb::EdgeStateBounds](#) = BoundConstraint< BOUND\_TYPE::LOWERUPPER, StateVertex< p >, BOUND\_VARS::ALL >  
*Bound constraint edge for states:  $\mathbf{x}_{min} \leq \mathbf{x}_k \leq \mathbf{x}_{max}$ .*

## 17.12 config.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <stdint>
```

## Classes

- class [teb::Config](#)  
*Configurations for Controller and Solver classes.*
- struct [teb::Config::Teb](#)  
*Configurations related to the TEB algorithm.*
- struct [teb::Config::Optim](#)  
*Configurations related to the trajectory optimization.*
- struct [teb::Config::Optim::Solver](#)  
*Configurations related to solvers.*
- struct [teb::Config::Optim::Solver::Lsq](#)  
*Configurations especially for least-squares-solvers.*
- struct [teb::Config::Optim::Solver::NonlinearProgram](#)  
*Settings for constrained optimization solver (nonlinear program solver)*
- struct [teb::Config::Optim::Solver::NonlinearProgram::Hessian](#)  
*Configurations for the hessian of the lagrangian calculation.*
- struct [teb::Config::Optim::Solver::NonlinearProgram::LineSearch](#)  
*Settings for the line-search algorithm (force merit function descent)*

- struct [teb::Config::Optim::Solver::Nlopt](#)  
*Configurations especially for the nlopt solver wrapper.*
- struct [teb::Config::Utilities](#)  
*Configurations for helper miscellaneous and utilities.*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## Enumerations

- enum [teb::FiniteDifferences](#) { [teb::FiniteDifferences::FORWARD](#), [teb::FiniteDifferences::CENTRAL](#) }  
*Enumeration for different finite difference types used especially for dynamic system discretization.*
- enum [teb::HessianMethod](#) { [teb::HessianMethod::NUMERIC](#), [teb::HessianMethod::BLOCK\\_BFGS](#), [teb::HessianMethod::FULL\\_BFGS](#), [teb::HessianMethod::FULL\\_BFGS\\_WITH\\_STRUCTURE\\_FILTER](#), [teb::HessianMethod::ZERO\\_HESSIAN](#) }  
*Enumeration for different hessian calculation methods (not necessary for least-squares solver)*
- enum [teb::HessianInit](#) { [teb::HessianInit::ZERO](#), [teb::HessianInit::IDENTITY](#), [teb::HessianInit::NUMERIC](#) }  
*Enumeration for the hessian initialization (relevant for BFGS hessian approximations)*
- enum [teb::NloptAlgorithms](#) { [teb::NloptAlgorithms::SLSQP](#) }  
*Enumeration for different solver algorithms supported by the Nlopt library (only those that are working here)*

## 17.13 download.md File Reference

## 17.14 graph.h File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/workspaces.h>
#include <Eigen/Core>
#include <stack>
#include <vector>
#include <memory>
```

## Classes

- class [teb::VertexType](#)  
*Generic interface class for vertices.*
- class [teb::EdgeType](#)  
*Generic interface class for edges.*
- class [teb::HyperGraph](#)  
*Graph object that stores pointers to active vertices and edges.*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## Enumerations

- enum [teb::FUNCT\\_TYPE](#) {  
[teb::FUNCT\\_TYPE::NONLINEAR](#), [teb::FUNCT\\_TYPE::NONLINEAR\\_SQUARED](#), [teb::FUNCT\\_TYPE::NONLINEAR\\_ONCE\\_DIFF](#), [teb::FUNCT\\_TYPE::QUADRATIC](#),  
[teb::FUNCT\\_TYPE::LINEAR\\_SQUARED](#), [teb::FUNCT\\_TYPE::LINEAR](#) }

*Enum for different function types (see [EdgeType](#) class)*

## 17.15 groups.dox File Reference

## 17.16 install.md File Reference

## 17.17 integrators.h File Reference

```
#include <teb_package/base/system_dynamics.h>
#include <Eigen/Core>
#include <queue>
```

## Classes

- class [teb::NumericalIntegrator< p, q >](#)  
*Interface for numerical integration methods used in simulation.*
- class [teb::ExplicitEuler< p, q >](#)  
*Simple explicit euler method for integration.*
- class [teb::RungeKuttaClassic< p, q >](#)  
*Classic Runge Kutta method (fourth order)*
- class [teb::RungeKutta5thOrder< p, q >](#)  
*Runge Kutta method (fifth order, slightly modified)*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## Enumerations

- enum [teb::NumericalIntegrators](#) { [teb::NumericalIntegrators::EXPLICIT\\_EULER](#), [teb::NumericalIntegrators::RUNGE\\_KUTTA\\_CLASSIC](#), [teb::NumericalIntegrators::RUNGE\\_KUTTA\\_5TH](#) }

*Enumeration for different numerical integration methods.*

## 17.18 mainpage.dox File Reference

## 17.19 matlab\_class\_handle.hpp File Reference

## 17.20 matlab\_interface.md File Reference

## 17.21 `measure_cpu_time.h` File Reference

```
#include <teb_package/base/typedefs.h>
#include <chrono>
```

### Macros

- `#define START_TIMER` auto start = std::chrono::high\_resolution\_clock::now();
- `#define STOP_TIMER(name)`

### 17.21.1 Macro Definition Documentation

17.21.1.1 `#define START_TIMER` auto start = std::chrono::high\_resolution\_clock::now();

Examples:

[integrator\\_system1.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_state\\_space.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Definition at line 12 of file `measure_cpu_time.h`.

17.21.1.2 `#define STOP_TIMER( name )`

Value:

```
PRINT_INFO("RUNTIME of " << name << ": " << \
    std::chrono::duration_cast<std::chrono::milliseconds>( \
        std::chrono::high_resolution_clock::now()-start \
    ).count() << " ms " << "(Compiled in Debug mode)");
```

Examples:

[integrator\\_system1.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_state\\_space.cpp](#), [mobile\\_robot\\_teb.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Definition at line 19 of file `measure_cpu_time.h`.

## 17.22 `misc.h` File Reference

```
#include <teb_package/base/typedefs.h>
#include <math.h>
#include <functional>
#include <tuple>
```

### Functions

- double `norm_angle` (double angle)  
*Normalize angle to interval  $[-\pi, \pi]$*
- void `norm_angle_vec` (Eigen::Ref< Eigen::MatrixXd > angles)  
*Normalize vector or matrix of angles to interval  $[-\pi, \pi]$*



### 17.22.1 Function Documentation

17.22.1.1 `double norm_angle ( double angle )` `[inline]`

## Parameters

<i>angle</i>	angle in radiant
--------------	------------------

## Returns

normalized angle to  $[-\pi, \pi]$  in rad

## Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 15 of file misc.h.

References PI.

Referenced by norm\_angle\_vec().

17.22.1.2 void norm\_angle\_vec ( Eigen::Ref< Eigen::MatrixXd > *angles* ) [inline]

## Parameters

<i>angles</i>	angle vector/matrix in radiant [all values will be replaced by its normalized version]
---------------	--

## Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 35 of file misc.h.

References norm\_angle().

## 17.23 simulator.h File Reference

```
#include <teb_package/base/base_controller.h>
#include <teb_package/base/system_dynamics.h>
#include <teb_package/simulation/integrators.h>
#include <teb_package/visualization/teb_plotter.h>
#include <functional>
#include <teb_package/simulation/simulator.hpp>
```

## Classes

- class [teb::SimResults](#)  
*Simulation results are stored and combined in this container class.*
- struct [teb::SimResults::TimeSeries](#)  
*Store measurements of dynamic systems (states and control inputs) w.r.t.*
- class [teb::Simulator< p, q >](#)  
*Simulator for dynamic systems controlled by a [TebController](#).*

## Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.24 simulator.hpp File Reference

```
#include <teb_package/simulation/simulator.h>
```

### Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.25 solver\_levenbergmarquardt\_eigen\_dense.cpp File Reference

```
#include <teb_package/solver/solver_levenbergmarquardt_eigen_dense.h>
#include <iostream>
```

### Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.26 solver\_levenbergmarquardt\_eigen\_dense.h File Reference

```
#include <teb_package/base/base_solver_least_squares.h>
```

### Classes

- class [teb::SolverLevenbergMarquardtEigenDense](#)

*Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Dense matrices version).*

### Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.27 solver\_levenbergmarquardt\_eigen\_sparse.cpp File Reference

```
#include <teb_package/solver/solver_levenbergmarquardt_eigen_sparse.h>
#include <iostream>
```

### Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.28 solver\_levenbergmarquardt\_eigen\_sparse.h File Reference

```
#include <teb_package/base/base_solver_least_squares.h>
#include <Eigen/Sparse>
```

### Classes

- class [teb::SolverLevenbergMarquardtEigenSparse](#)  
*Levenberg-Marquardt Solver that uses Eigen to solve the resulting linear system (Sparse matrices version).*

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.29 solver\_nlopt\_package.cpp File Reference

## 17.30 solver\_nlopt\_package.h File Reference

```
#include <teb_package/base/base_solver.h>
```

### Classes

- class [teb::SolverNloptPackage](#)  
*This class wraps the optimization problem to allow solving by NLOPT.*

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.31 solver\_overview.md File Reference

## 17.32 solver\_sqp\_dense.h File Reference

```
#include <teb_package/solver/base_solver_nonlinear_program_dense.h>
#include <qpOASES.hpp>
```

### Classes

- class [teb::SolverSQPDense](#)  
*Dense sequential quadratic programming (SQP) solver for nonlinear programs.*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## Macros

- `#define __SUPPRESSANYOUTPUT__`

### 17.32.1 Macro Definition Documentation

#### 17.32.1.1 `#define __SUPPRESSANYOUTPUT__`

Definition at line 6 of file solver\_sqp\_dense.h.

## 17.33 solver\_sqp\_dense\_backup.h File Reference

```
#include <teb_package/solver/base_solver_nonlinear_program_dense.h>
#include <qpOASES.hpp>
```

## Classes

- class [teb::SolverSQPDense](#)

*Dense sequential quadratic programming (SQP) solver for nonlinear programs.*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.34 system\_dynamics.h File Reference

```
#include <Eigen/Core>
#include <Eigen/Dense>
```

## Classes

- class [teb::SystemDynamics< p, q >](#)

*Helper class for modeling nonlinear dynamic systems.*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.35 `teb_controller.h` File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/base_controller.h>
#include <teb_package/base/teb_vertices.h>
#include <teb_package/base/common_teb_edges.h>
#include <teb_package/base/base_solver.h>
#include <Eigen/Core>
#include <Eigen/StdDeque>
#include "algorithm"
#include <teb_package/base/teb_controller.hpp>
```

### Classes

- class [teb::TebController](#)< [p](#), [q](#) >  
*Main Timed-Elastic-Band controller class.*

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.36 `teb_controller.hpp` File Reference

```
#include <teb_package/base/teb_controller.h>
```

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.37 `teb_plotter.cpp` File Reference

```
#include <teb_package/visualization/teb_plotter.h>
```

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.38 `teb_plotter.h` File Reference

```
#include <teb_package/base/typedefs.h>
#include <teb_package/base/teb_controller.h>
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <teb_package/visualization/teb_plotter.hpp>
```

### Classes

- struct [teb::PlotOptions](#)  
*Customize plots and figures.*
- class [teb::TebPlotter](#)  
*Provides useful visualization and plotting functions.*

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.39 `teb_plotter.hpp` File Reference

```
#include <teb_package/visualization/teb_plotter.h>
```

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*

## 17.40 `teb_vertices.h` File Reference

```
#include <teb_package/base/graph.h>
```

### Classes

- class [teb::StateVertex< p >](#)  
*Vertex that contains the StateVector.*
- class [teb::ControlVertex< q >](#)  
*Vertex that contains the ControlVector.*
- class [teb::TimeDiff](#)  
*Vertex that contains the time information of the TEB:  $\Delta T$ .*

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## 17.41 typedefs.h File Reference

```
#include <limits.h>
#include <iostream>
#include <vector>
#include <Eigen/Core>
#include <Eigen/StdVector>
#include <stack>
#include <deque>
```

## Namespaces

- [teb](#)

*General project namespace for all library functions and objects.*

## Macros

- `#define PI 3.14159265358979323846264`  
*Define  $\pi$ .*
- `#define INF HUGE_VAL`  
*Define Infinity  $\infty$ .*
- `#define PRINT\_DEBUG(msg) std::cout << "Debug: " << msg << std::endl;`  
*Print msg-stream only if project is compiled in Debug-mode.*
- `#define PRINT\_DEBUG\_ONCE(msg) { static const auto debugOnce = [&] { std::cout << "Debug: " << msg << std::endl; return true;}(); (void)debugOnce; }`  
*Print msg-stream only once and only if project is compiled in Debug-mode.*
- `#define PRINT\_DEBUG\_COND(cond, msg) if (cond) std::cout << "Debug: " << msg << std::endl;`  
*Print msg-stream only if cond == true and only if project is compiled in Debug-mode.*
- `#define PRINT\_DEBUG\_COND\_ONCE(cond, msg) { static const auto debugOnce = [&] { if (cond) std::cout << "Debug: " << msg << std::endl; return true;}(); (void)debugOnce; }`  
*Print msg-stream only if cond == true, only once and only if project is compiled in Debug-mode.*
- `#define PRINT\_INFO(msg) std::cout << "Info: " << msg << std::endl;`  
*Print msg-stream.*
- `#define PRINT\_INFO\_ONCE(msg) { static const auto infoOnce = [&] { std::cout << "Info: " << msg << std::endl; return true;}(); (void)infoOnce; }`  
*Print msg-stream only once.*
- `#define PRINT\_INFO\_COND(cond, msg) if (cond) std::cout << "Info: " << msg << std::endl;`  
*Print msg-stream only if cond == true.*
- `#define PRINT\_INFO\_COND\_ONCE(cond, msg) { static const auto infoOnce = [&] { if (cond) std::cout << "Info: " << msg << std::endl; return true;}(); (void)infoOnce; }`  
*Print msg-stream only if cond == true, only once.*
- `#define PRINT\_ERROR(msg) std::cerr << "Error: " << msg << std::endl;`  
*Print msg-stream as error msg.*
- `#define INPUT\_STREAM(variable, default_val) std::cin >> variable;`



## Typedefs

- `template<typename T >`  
`using teb::EigenScalar = Eigen::Matrix< T, 1, 1 >`  
*Define 1x1 Eigen::Matrix.*
- `using teb::EigenScalarD = EigenScalar< double >`  
*Define 1x1 Eigen::Matrix of type double.*
- `template<typename T >`  
`using teb::BackupStackType = std::stack< T, std::deque< T > >`  
*Backup stack type.*

### 17.41.1 Macro Definition Documentation

#### 17.41.1.1 `#define INF HUGE_VAL`

Definition at line 39 of file typedefs.h.

Referenced by `teb::BoundConstraint< Bound_type, Vertex, Bound_vars, Index >::getCustomData()`, and `teb::SolverSQPDense::initSolverWorkspace()`.

#### 17.41.1.2 `#define INPUT_STREAM( variable, default_val ) std::cin >> variable;`

Definition at line 105 of file typedefs.h.

Referenced by `teb::Simulator< p, q >::simClosedLoop()`.

#### 17.41.1.3 `#define PI 3.14159265358979323846264`

Definition at line 36 of file typedefs.h.

Referenced by `norm_angle()`.

#### 17.41.1.4 `#define PRINT_DEBUG( msg ) std::cout << "Debug: " << msg << std::endl;`

Definition at line 63 of file typedefs.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessianFullBFGS()`, `teb::TebPlotter::setOutputToFile()`, and `teb::SolverSQPDense::solveImpl()`.

#### 17.41.1.5 `#define PRINT_DEBUG_COND( cond, msg ) if (cond) std::cout << "Debug: " << msg << std::endl;`

Definition at line 69 of file typedefs.h.

#### 17.41.1.6 `#define PRINT_DEBUG_COND_ONCE( cond, msg ) { static const auto debugOnce = [&] { if (cond) std::cout << "Debug: " << msg << std::endl; return true; }(); (void)debugOnce; }`

Definition at line 72 of file typedefs.h.

Referenced by `teb::TebController< p, q >::initOptimization()`, `teb::BaseSolverLeastSquares::initWorkspaces()`, and `teb::BaseSolverNonlinearProgramDense::initWorkspaces()`.

```
17.41.1.7 #define PRINT_DEBUG_ONCE( msg ) { static const auto debugOnce = [&] { std::cout << "Debug: " << msg <<
std::endl; return true; }(); (void)debugOnce; }
```

Definition at line 66 of file typedefs.h.

Referenced by `teb::TebPlotter::setOutputToFile()`, and `teb::SolverSQPDense::solveImpl()`.

```
17.41.1.8 #define PRINT_ERROR( msg ) std::cerr << "Error: " << msg << std::endl;
```

Definition at line 103 of file typedefs.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, and `teb::BaseSolverNonlinearProgramDense::initHessianBFGS()`.

```
17.41.1.9 #define PRINT_INFO( msg ) std::cout << "Info: " << msg << std::endl;
```

Examples:

[integrator\\_system1.cpp](#), [integrator\\_system\\_classic\\_mpc.cpp](#), [linear\\_system\\_ode.cpp](#), [linear\\_system\\_state\\_space.cpp](#), and [van\\_der\\_pol\\_system.cpp](#).

Definition at line 91 of file typedefs.h.

Referenced by `teb::BaseSolverNonlinearProgramDense::calculateLagrangianHessian()`, `teb::TebPlotter::plot()`, `teb::TebPlotter::plot1DVector()`, `teb::Simulator< p, q >::plotResults()`, `teb::TebPlotter::plotTEB()`, `teb::TebPlotter::plotTwoCol()`, `teb::Simulator< p, q >::simClosedLoop()`, `teb::TebPlotter::spyMatrix()`, and `teb::TebPlotter::TebPlotter()`.

```
17.41.1.10 #define PRINT_INFO_COND( cond, msg ) if (cond) std::cout << "Info: " << msg << std::endl;
```

Definition at line 97 of file typedefs.h.

```
17.41.1.11 #define PRINT_INFO_COND_ONCE( cond, msg ) { static const auto infoOnce = [&] { if (cond) std::cout << "Info: "
<< msg << std::endl; return true; }(); (void)infoOnce; }
```

Definition at line 100 of file typedefs.h.

```
17.41.1.12 #define PRINT_INFO_ONCE( msg ) { static const auto infoOnce = [&] { std::cout << "Info: " << msg <<
std::endl; return true; }(); (void)infoOnce; }
```

Examples:

[mobile\\_robot\\_teb.cpp](#).

Definition at line 94 of file typedefs.h.

Referenced by `teb::Simulator< p, q >::simClosedLoop()`.

## 17.42 utilities.h File Reference

```
#include "misc.h"
#include "measure_cpu_time.h"
```

## 17.43 workspaces.h File Reference

```
#include <Eigen/Core>
#include <Eigen/StdVector>
#include <vector>
```

### Classes

- class [teb::JacobianWorkspace](#)  
*Memory management and accessor functions for the Block-Jacobians (for each edge).*
- class [teb::HessianWorkspace](#)  
*Memory management and accessor functions for the Block-Hessian (for each edge).*

### Namespaces

- [teb](#)  
*General project namespace for all library functions and objects.*



## Chapter 18

# Example Documentation

### 18.1 integrator\_system1.cpp

The following example shows the TEB controller applied to an integrator system with  $p$  integrator states:

$$Tx^{(p)} = u$$

The control input  $u$  should be bounded to the interval  $[-1, 1]$ .

Considering the particular case  $p = 4$ . The state vector is defined by  $\mathbf{x} = [x, \dot{x}, \ddot{x}, \dddot{x}]^T$ . Transforming ODE (1) into a state space corresponding to  $\mathbf{x}$  leads to:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \\ \ddddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1/T \end{bmatrix} u$$

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0, 0, 0]^T$  to the final state  $\mathbf{x}_f = [1, 0, 0, 0]^T$ .

The system dynamics are specified using the IntegratorSystem class. See the following header file how to add the state space model mentioned above for an integrator with  $p$  states.

File **integrator\_system.h**:

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__

#include <teb_package.h>

template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        const int no_int = p-1;
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

        // add last equation containing control u
        state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
        return state_equations;
    }
}
```

```

void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:
    double _time_constant = 1;
};

#endif /* defined(__teb_package__examples_integrator_system__) */

```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the IntegratorSystem object specified in integrator\_system.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

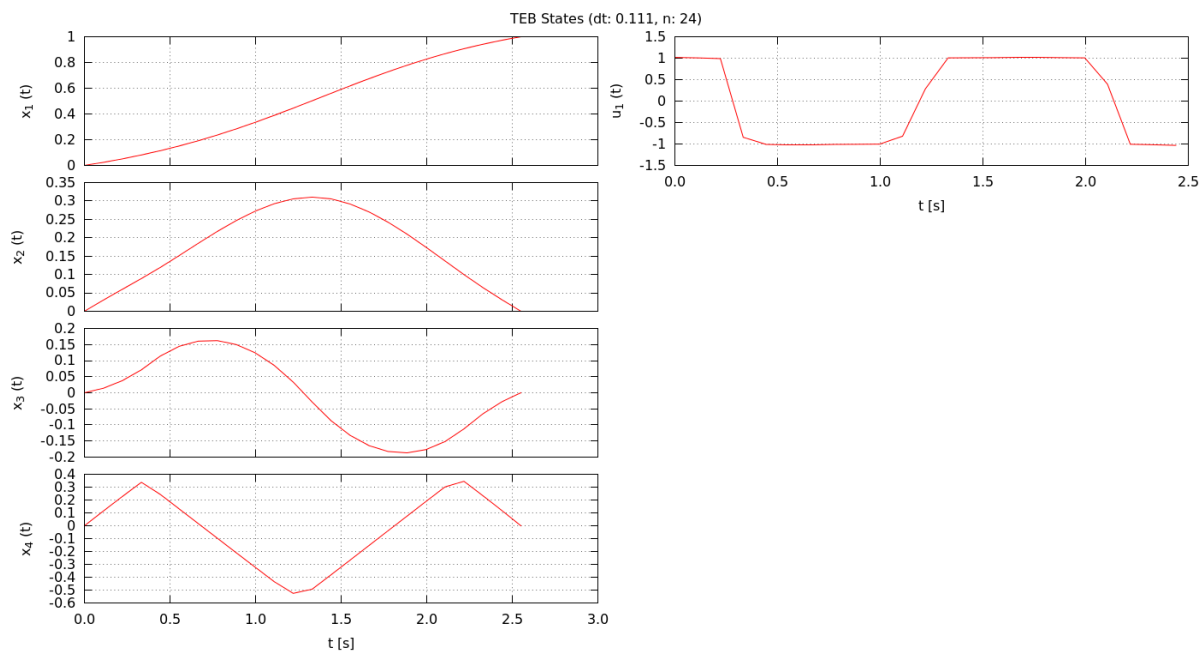


Figure 18.1: Result of integrator\_system1.cpp

### File integrator\_system1.cpp

```

#include "integrator_system.h"

int main()
{
    const int p = 4;
    double x0[] = {0,0,0,0};
    double xf[] = {1,0,0,0};
    double u = 0;

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;

    // test SQP solver
    teb::Config cfg;
    //cfg.teb.teb_iter = 1;
    //cfg.optim.solver.nonlin_prog.hessian.hessian_method = teb::HessianMethod::FULL_BFGS;
    //cfg.optim.solver.nonlin_prog.hessian.hessian_init = teb::HessianInit::IDENTITY;
    //cfg.optim.solver.nonlin_prog.hessian.hessian_init_identity_scale = 0.5;
    //teb::SolverSQPDense solver;

    teb::TebController<p,1> teb(&cfg, &solver);
}

```

```

// System specific settings
IntegratorSystem<p> system;
system.setTimeConstant(1.0);

teb.setSystemDynamics(&system);

teb.activateObjectiveTimeOptimal();
teb.activateControlBounds(0, -1.0, 1.0);

// Perform open-loop planning
START_TIMER;
teb.step(x0, xf, &u);
STOP_TIMER("TEB optimization loop");

// Print determined control for the current sampling interval
PRINT_INFO("Control u: " << u);

// Plot states and control input
teb::TebPlotter plotter;
// optional: plotter.setOutputToFile("teb_opt_int",teb::TebPlotter::FileFormat::PNG);
plotter.plotTEB(teb);

return 0;
}

```

## 18.2 integrator\_system2.cpp

The following example shows the TEB controller applied to an integrator system with  $p$  integrator states:

$$Tx^{(p)} = u$$

The control input  $u$  should be bounded to the interval  $[-1, 1]$ .

Considering the particular case  $p = 2$ . The state vector is defined by  $\mathbf{x} = [x, \dot{x}]^T$ . Transforming ODE (1) into a state space corresponding to  $\mathbf{x}$  leads to:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 1/T \end{bmatrix} u$$

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0]^T$  to the final state  $\mathbf{x}_f = [1, 0]^T$ .

The system dynamics are specified using the IntegratorSystem class. See the following header file how to add the state space model mentioned above for an integrator with  $p$  states.

File **integrator\_system.h**:

```

#ifndef __teb_package_examples_integrator_system__
#define __teb_package_examples_integrator_system__

#include <teb_package.h>

template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        const int no_int = p-1;
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

        // add last equation containing control u
        state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
        return state_equations;
    }

    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

```

```
protected:

    double _time_constant = 1;
};

#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the IntegratorSystem object specified in integrator\_system.h.

Additionally, a Simulator object is initialized in order to perform closed-loop and open-loop simulations.

The example program compares the result for the open-loop control and the simulated closed-loop control. In addition the first optimization result (red) is shown. If the system dynamic equation (equality constraint) is fully satisfied, red and green should be similar. They can only be identical, if the integrator method NumericalIntegrators-:ForwardEuler is used in simulation, since the TEB relies on the very same integration model.

The output of the program is as follows:

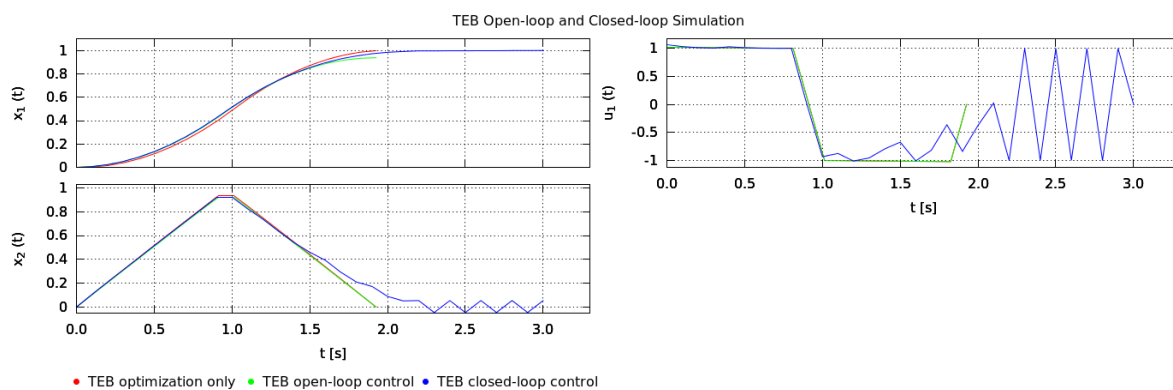


Figure 18.2: Result of integrator\_system2.cpp

### File integrator\_system2.cpp

```
#include "integrator_system.h"

int main()
{
    const int p = 2; // number of states
    double x0[] = {0,0}; // start state
    double xf[] = {1,0}; // goal state

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);

    // System specific settings
    IntegratorSystem<p> system;
    teb.setSystemDynamics(&system);

    // Optional: smooth control input trajectory
    // with small weighted quadratic cost
    //teb.activateObjectiveQuadraticForm(0,0.1,0);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(&teb,system,&plotter);
    // Set simulation sample time
    sim.setSampleTime(0.1);
    // Simulate 3 seconds
    sim.simOpenAndClosedLoop(x0,xf,3);
```



```
    return 0;
}
```

## 18.3 integrator\_system\_classic\_mpc.cpp

The following example shows a classic MPC controller (with a quadratic cost functional and a receding horizon) applied to an integrator system with  $p$  integrator states:

$$Tx^{(p)} = u$$

The control input  $u$  should be bounded to the interval  $[-1, 1]$ .

Considering the particular case  $p = 3$ . The state vector is defined by  $\mathbf{x} = [x, \dot{x}, \ddot{x}]^T$ . Transforming ODE (1) into a state space corresponding to  $\mathbf{x}$  leads to:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \ddot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1/T \end{bmatrix} u$$

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0]^T$  to the final state  $\mathbf{x}_f = [1, 0]^T$  while minimizing the quadratic control effort and control energy:

$$J = (\mathbf{x}_n - \mathbf{x}_f)^T \mathbf{Q}_f (\mathbf{x}_n - \mathbf{x}_f) + \sum_{k=0}^{n-1} [(\mathbf{x}_k - \mathbf{x}_f)^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_f) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k]$$

The receding/moving horizon is set to  $n = 10$ . The system dynamics are descretized using finite differences and with a uniform and fixed resolution ( $\Delta T = 0.1\text{s}$ ).

The continuous system dynamics are specified using the IntegratorSystem class. See the following header file how to add the state space model mentioned above for an integrator with  $p$  states.

File **integrator\_system.h**:

```
#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__

#include <teb_package.h>

template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        const int no_int = p-1;
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

        // add last equation containing control u
        state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
        return state_equations;
    }

    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:
    double _time_constant = 1;
};

#endif /* defined(__teb_package__examples_integrator_system__) */
```

The following cpp-file shows how to create the `TebController` object with common MPC characteristics, how to add control bounds and how to add the `IntegratorSystem` object specified in `integrator_system.h`.

Additionally, a `Simulator` object is initialized in order to perform closed-loop and open-loop simulations.

The example program compares the result for the open-loop control and the simulated closed-loop control. In addition the first optimization result (red) is shown (the first horizon!).

The output of the program is as follows:

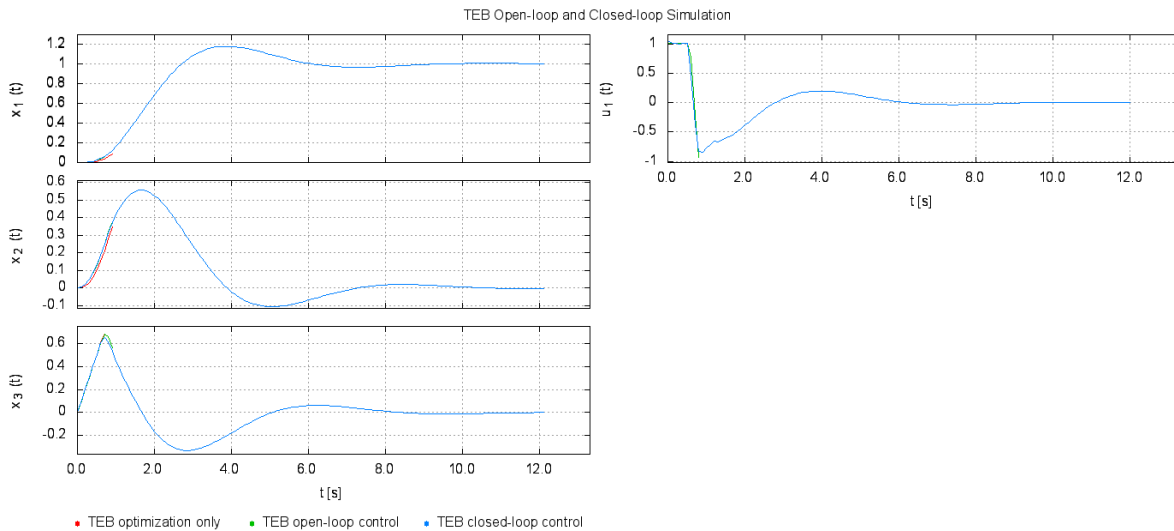


Figure 18.3: Result of `integrator_system_classic_mpc.cpp`

#### File `integrator_system_classic_mpc.cpp`

```
#include "integrator_system.h"

int main()
{
    const int p = 3;
    double x0[] = {0,0,0};
    double xf[] = {1,0,0};
    //double u; //! store control

    // Setup soft constraint weights
    teb::Config cfg;
    cfg.optim.solver.lsqr.weight_equalities = 2;
    cfg.optim.solver.lsqr.weight_inequalities = 2;
    cfg.optim.solver.lsqr.weight_adaptation_factor = 5;

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&cfg, &solver);

    // Setup receding horizon T = n * dt = 10 * 0.1
    // Common setting: unfixed goal and fixed resolution
    teb.setupHorizon(false, true, 10, 0.1);

    // Setup optimization problem
    teb.activateObjectiveQuadraticForm(0.5,0.1,50); // parameters: Q, R, Qf
    teb.activateControlBounds(0, -1.0, 1.0);

    IntegratorSystem<p> system;
    system.setTimeConstant(1.0);

    teb.setSystemDynamics(&system);

    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(&teb, system, &plotter);

    // Set simulation sample time
```

```

sim.setSampleTime(0.1);

// Simulate 12 seconds and save image to file (look in the folder of the binary)
plotter.setOutputToFile("mpc_control_integrator",
    teb::TebPlotter::FileFormat::PNG);

PRINT_INFO("Simulation running...");
sim.simOpenAndClosedLoop(x0, xf, 12, false, false);
PRINT_INFO("Simulation finished. Find the output file in your current working folder.");

return 0;
}

```

## 18.4 integrator\_system\_mex.cpp

The following examples clarifies, how to prepare the TEB Controller (in particular the Controller of the Integrator-System example) for the usage in Matlab as a mex function.

The header file remains unchanged. File `integrator_system.h`:

```

#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__

#include <teb_package.h>

template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        const int no_int = p-1;
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

        // add last equation containing control u
        state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
        return state_equations;
    }

    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:
    double _time_constant = 1;
};

#endif /* defined(__teb_package__examples_integrator_system__) */

```

The following cpp-file shows how embed the TebController object into Matlab's function wrapper `mexFunction`. The example also wraps the mex function into a Matlab class. The controller class object is transformed to a Matlab handle using a small help (undocumented) helper class `teb::matlab::matlabClassHandle()`. See the Matlab class `control_interface.cpp` in `examples/matlab_interface` directory for more details on how to use it in Matlab.

An alternate approach without the helper class would be to store a persistent object (and its pointer) within the file (like for the Solver and System class). **Note**, the wrapper class can be easily adapted to other controllers by just changing the typedefs at the top of the file. Controller settings can be modified below the comment *// Setup Controller*.

File `integrator_system_mex.cpp`

```

#include "mex.h"
#include <iostream>

#include "integrator_system.h"

```

```

// support helper class to define a pointer suited for matlabs handle object
// @see teb::matlab::matlabClassHandle
using namespace teb::matlab;

// Create typedef for our controller, solver and system type
typedef teb::TebController<2, 1> ControllerType;
typedef teb::SolverLevenbergMarquardtEigenSparse SolverType;
typedef IntegratorSystem<2> SystemType;

// Create global variables
std::unique_ptr<SolverType> solver; // Solver
std::unique_ptr<SystemType> dynamics; // System dynamics

void mexFunction(int nlhs, mxArray *plhs[],
                 int nrhs, const mxArray *prhs[])
{
    // Get command string
    char cmd[64];
    if (nrhs < 1 || mxGetString(prhs[0], cmd, sizeof(cmd)))
        mexErrMsgTxt("First input should be a command string with less than 64 characters.");

    // Create new object if command equals "new"
    if (!strcmp("new", cmd))
    {
        // Check parameters
        if (nlhs != 1)
            mexErrMsgTxt("New: One output expected.");

        // Instantiate new objects
        solver = std::unique_ptr<SolverType>(new SolverType);
        dynamics = std::unique_ptr<SystemType>(new SystemType);
        ControllerType* ctrl = new ControllerType(solver.get());

        // Setup controller
        ctrl->setSystemDynamics(dynamics.get()); // Satisfy system dynamics

        ctrl->activateObjectiveTimeOptimal(); // Minimum time control
        ctrl->activateControlBounds(0, -1.0, 1.0); // Control input bounds

        // Return handle to the new controller instance
        plhs[0] = convertPtr2Mat< ControllerType >(ctrl);

        return;
    }

    // For further work, we need a second input (class instance handle)
    if (nrhs < 2)
        mexErrMsgTxt("Second input should be a class instance handle.");

    // Delete object
    if (!strcmp("delete", cmd))
    {
        destroyObject<ControllerType>(prhs[1]);
        if (nlhs != 0 || nrhs != 2)
            mexWarnMsgTxt("Delete: Unexpected arguments ignored.");
        return;
    }

    // For further work, retrieve the class instance pointer from the second input
    ControllerType* ctrl = convertMat2Ptr<ControllerType>(prhs[1]);

    // Now start calling desired class method

    // Call teb::BaseController::step()
    if (!strcmp("step", cmd))
    {
        // Check inputs and outputs
        if (nrhs != 4) mexErrMsgIdAndTxt("teb_package:step:nrhs", "Four inputs required: cmd, obj, x0 and xf [2x1].");
        if (nlhs != 1) mexErrMsgIdAndTxt("teb_package:step:nlhs", "One output required: u [1x1]//"); //
        teb [3 x n_max], dt [1x1].");

        double* x0 = mxGetPr(prhs[2]); // start state
        double* xf = mxGetPr(prhs[3]); // goal state
        double u = 0; // store control

        // Perform open-loop planning
        ctrl->step(x0, xf, &u);

        // create output (control input u)
        plhs[0] = mxCreateDoubleScalar(u);
        return;
    }
}

```

```

// Return TEB matrix teb::BaseController::getStateCtrlInfoMat()
if (!strcmp("tebmatrix", cmd))
{
    // Check inputs and outputs
    if (nrhs != 2) mexErrMsgTxt("No more input parameter allowed.");
    if (nlhs != 1) mexErrMsgTxt("One output for the teb matrix required.");

    // Size of the mat: [p+q+1 x n] // +1: add time vector
    int m = ctrl->NoStates + ctrl->NoControls + 1;
    int n = ctrl->getN();
    plhs[0] = mxCreateNumericMatrix(m, n, mxDOUBLE_CLASS, mxREAL);

    // Get values
    // Wrap mxArray into an Eigen::Matrix
    Eigen::Map<Eigen::Matrix<double, -1, -1, Eigen::ColMajor>> mat_map(mxGetPr(plhs[0]), m, n); //
    Matlab uses column major representation
    mat_map.topRows(1) = ctrl->getAbsoluteTimeVec().transpose();
    mat_map.bottomRows(m-1) = ctrl->getStateCtrlInfoMat();
    return;
}

// If this line is reached, the command is not recognized
mexErrMsgTxt("Command noch recognized.");
}

```

## 18.5 integrator\_system\_sfuns.cpp

The following examples clarify, how to prepare the TEB Controller (in particular the Controller of the Integrator-System example) for the usage in Matlab Simulink as a S-Function block.

The header file remains unchanged. File **integrator\_system.h**:

```

#ifndef __teb_package__examples_integrator_system__
#define __teb_package__examples_integrator_system__

#include <teb_package.h>

template <int p>
class IntegratorSystem : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        const int no_int = p-1;
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

        // add last equation containing control u
        state_equations[no_int] = u[0] / _time_constant; // x^(n) = u / T
        return state_equations;
    }

    void setTimeConstant(double time_constant) {_time_constant = time_constant;};

protected:
    double _time_constant = 1;
};

#endif /* defined(__teb_package__examples_integrator_system__) */

```

The following cpp-file shows how embed the TebController object into Matlab's S-Function wrappers.

File **integrator\_system\_sfuns.cpp**

```

#include "integrator_system.h"

```

```

#ifdef __cplusplus
extern "C" { // use the C fcn-call standard for all functions
#endif // defined within this scope

#define S_FUNCTION_NAME integrator_system_sfun /* Defines and Includes */
#define S_FUNCTION_LEVEL 2

/*
 * Need to include simstruc.h for the definition of the SimStruct and
 * its associated macro definitions.
 */
#include "simstruc.h"

#define MDL_INITIAL_SIZES
static void mdlInitializeSizes(SimStruct *S)
{
    // Reserve elements in the pointer work vector to store C++ objects
    // (see mdlStart function for object instantiation)
    ssSetNumPWork(S, 3); // Assume 3 objects (controller, solver, system)

    // Check inputs and outputs
    ssSetNumSFcnParams(S, 0);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
        return; /* Parameter mismatch reported by the Simulink engine*/
    }

    if (!ssSetNumInputPorts(S, 2)) return;
    // input 1: xf
    ssSetInputPortWidth(S, 0, 2);
    ssSetInputPortDirectFeedThrough(S, 0, 1);

    // input 2: x0
    ssSetInputPortWidth(S, 1, 2);
    ssSetInputPortDirectFeedThrough(S, 1, 1);

    // output: u
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 1);

    ssSetNumSampleTimes(S, 1); // TODO

    /* Take care when specifying exception free code - see sfuntmpl.doc */
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

#define MDL_INITIALIZE_SAMPLE_TIMES
static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S, 0, INHERITED_SAMPLE_TIME); // TODO
    ssSetOffsetTime(S, 0, 0.0);
}

#define MDL_START
static void mdlStart(SimStruct *S)
{
    // Instantiate persistent objects (delete objects in mdlTerminate)
    teb::BaseSolver* solver = new
        teb::SolverLevenbergMarquardtEigenSparse;
    teb::TebController<2, 1>* ctrl = new
        teb::TebController<2, 1>(solver);
    IntegratorSystem<2>* int_sys = new IntegratorSystem<2>;

    // Store void pointers in simulinks work vector for persistent variables.
    ssGetPWork(S)[0] = (void*) solver; // solver
    ssGetPWork(S)[1] = (void*) ctrl; // controller
    ssGetPWork(S)[2] = (void*) int_sys; // system

    // setup controller
    ctrl->setSystemDynamics(int_sys);
    ctrl->activateObjectiveTimeOptimal();
    ctrl->activateControlBounds(0, -1.0, 1.0);
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
    // reset trajectory
    teb::TebController<2, 1>* ctrl = (
        teb::TebController<2, 1>*) ssGetPWork(S)[1];
    ctrl->resetController();
}

#define MDL_OUTPUTS
static void mdlOutputs(SimStruct *S, int_T tid)

```

```

{
    // Retrieve persistent controller object
    teb::TebController<2, 1>* ctrl = (
        teb::TebController<2, 1>*) ssGetPWork(S)[1];

    // Get input and output parameters
    InputRealPtrsType xfPtrs = ssGetInputPortRealSignalPtrs(S,0);
    InputRealPtrsType x0Ptrs = ssGetInputPortRealSignalPtrs(S,1);
    real_T *uPtr = ssGetOutputPortRealSignal(S,0);

    // Perform MPC step
    ctrl->step(*x0Ptrs, *xfPtrs, uPtr);
}

static void mdlTerminate(SimStruct *S)
{
    // Destroy persistent objects
    teb::BaseSolver* solver = (teb::BaseSolver*) ssGetPWork(S)[0];
    teb::TebController<2, 1>* ctrl = (
        teb::TebController<2, 1>*) ssGetPWork(S)[1];
    IntegratorSystem<2>* int_sys = (IntegratorSystem<2>*) ssGetPWork(S)[2];
    delete solver;
    delete ctrl;
    delete int_sys;
}

#ifdef MATLAB_MEX_FILE /* Is this file being compiled as a MEX-file? */
#include "simulink.c" /* MEX-file interface mechanism */
#else
#include "cg_sfun.h" /* Code generation registration function */
#endif

#ifdef __cplusplus
} // end of extern "C" scope
#endif

```

## 18.6 linear\_system\_ode.cpp

The following example shows the TEB controller applied to a common linear system described by an ordinary differential equation of the  $p$ -th order:

$$a_p x^{(p)}(t) + a_{p-1} x^{(p-1)}(t) + \dots + a_1 \dot{x}(t) + a_0 x(t) = bu(t)$$

The control input  $u$  should be bounded to the interval  $[-1, 1]$ .

Let  $p = 2$ ,  $b = 1$  and  $\mathbf{a} = [a_2, a_1, a_0] = [0.8, 0.9, 1.0]$  that leads to the following ode:

$$0.8\ddot{x}(t) + 0.9\dot{x}(t) + 1.0x(t) = u(t)$$

The state vector is defined by  $\mathbf{x} = [x, \dot{x}]^T$ .

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0]^T$  to the final state  $\mathbf{x}_f = [1, 0]^T$ .

The system dynamics are specified using the LinearSystemODE class. See the following header file how to add the state space model mentioned above.

File **linear\_system\_ode.h**:

```

#ifndef __teb_package__examples_linear_system_ode__
#define __teb_package__examples_linear_system_ode__

#include <teb_package.h>

template <int p>
class LinearSystemODE : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {

```

```

StateVector state_equations;

const int no_int = p-1;
// integrator states -> control normal form
if (p>1) state_equations.head(no_int) = x.segment(1,no_int); // x^(1:n-1) = x^(2:n)

// add last equation containing ODE coefficients
state_equations[no_int] = _coeff_b*u[0]; // u
for (int i = 0; i < p; ++i) // add all other ode coefficients
{
    state_equations[no_int] -= _coeff_a[p-i] * x[i];
}
state_equations[no_int] /= _coeff_a[0];
return state_equations;
}

void setODECoefficients(const double* coeff_a, double coeff_b)
{
    _coeff_a = coeff_a;
    _coeff_b = coeff_b;
}

protected:
    const double* _coeff_a = nullptr;
    double _coeff_b = 1;
};

#endif /* defined(__teb_package__examples_linear_system_ode__) */

```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the LinearSystemODE object specified in linear\_system\_ode.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

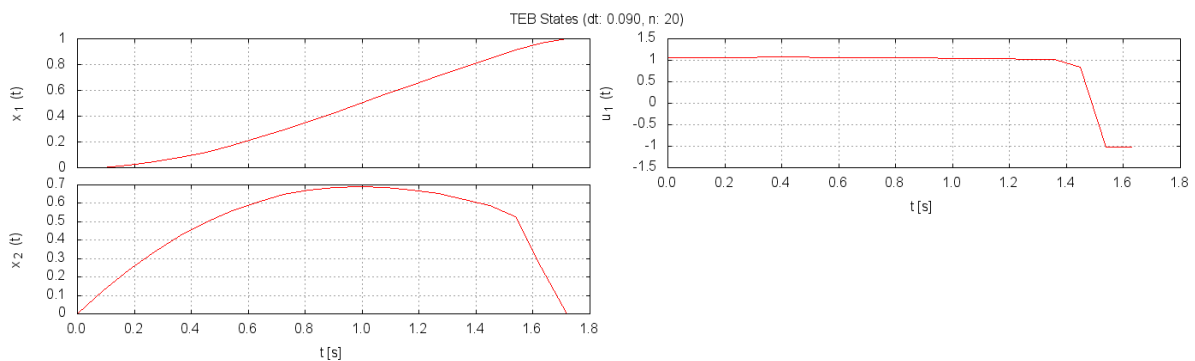


Figure 18.4: Result of linear\_system\_ode.cpp

#### File linear\_system\_ode.cpp

```

#include "linear_system_ode.h"

int main()
{
    const int p = 2; // number of states
    // ode: a_2 * xddot + a_1 * xdot + a_0 * x = b * u;
    const double coeff_a[] {0.8,0.9,1}; // a_2, a_1, a_0
    const double coeff_b = 1; // no derivatives of the input supported atm.

    double x0[] = {0,0}; // start state
    double xf[] = {1,0}; // goal state
    double u; // store control

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);
}

```



```

// System specific settings
LinearSystemODE<p> system;
system.setODECoefficients(coeff_a, coeff_b);

teb.setSystemDynamics(&system);

teb.activateObjectiveTimeOptimal();
teb.activateControlBounds(0, -1.0, 1.0);

// Perform open-loop planning
START_TIMER;
teb.step(x0, xf, &u);
STOP_TIMER("TEB optimization loop");

// Print determined control for the current sampling interval
PRINT_INFO("Control u: " << u);

// Plot states and control input
teb::TebPlotter plotter;
plotter.plotTEB(teb);

return 0;
}

```

## 18.7 linear\_system\_ode\_ctrl\_comparison.cpp

The following example compares the time-optimal TEB controller with the common quadratic form MPC.

The controlled system is specified using an (unstable) ordinary differential equation (refer to other examples for more details).

The system dynamics are specified using the LinearSystemODE class. See the following header file how to add the state space model mentioned above.

File **linear\_system\_ode.h**:

```

#ifndef __teb_package__examples_linear_system_ode__
#define __teb_package__examples_linear_system_ode__

#include <teb_package.h>

template <int p>
class LinearSystemODE : public teb::SystemDynamics<p,1>
{
public:
    using StateVector = typename teb::SystemDynamics<p,1>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,1>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        const int no_int = p-1;
        // integrator states -> control normal form
        if (p>1) state_equations.head(no_int) = x.segment(1,no_int); //  $x^{(1:n-1)} = x^{(2:n)}$ 

        // add last equation containing ODE coefficients
        state_equations[no_int] = _coeff_b*u[0]; // u
        for (int i = 0; i < p; ++i) // add all other ode coefficients
        {
            state_equations[no_int] -= _coeff_a[p-i] * x[i];
        }
        state_equations[no_int] /= _coeff_a[0];
        return state_equations;
    }

    void setODECoefficients(const double* coeff_a, double coeff_b)
    {
        _coeff_a = coeff_a;
        _coeff_b = coeff_b;
    }

protected:
    const double* _coeff_a = nullptr;
    double _coeff_b = 1;

```

```
};

#endif /* defined(__teb_package__examples_linear_system_ode__) */
```

The following cpp-file shows how to create both controllers, how to add control bounds and how to add the Linear-SystemODE object specified in linear\_system\_ode.h.

The output of the program is as follows (closed-loop results only):

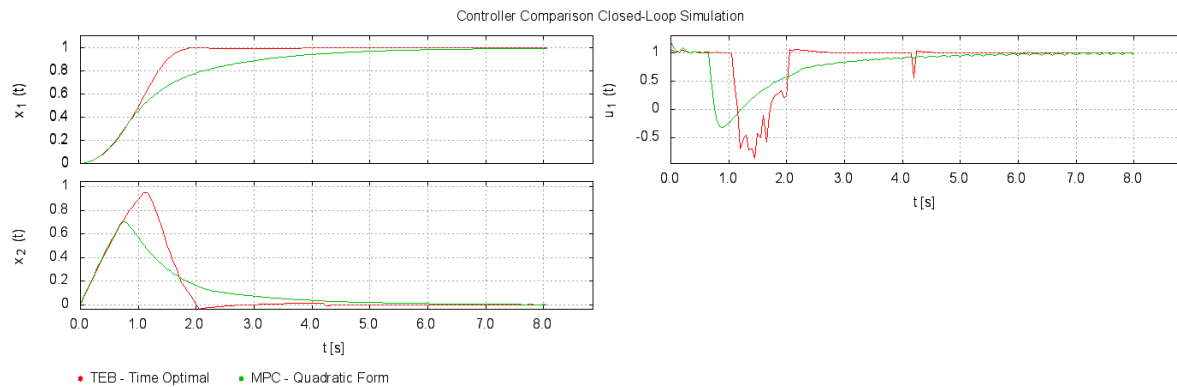


Figure 18.5: Result of linear\_system\_ode\_ctrl\_comparison.png

#### File linear\_system\_ode\_ctrl\_comparison.cpp

```
#include "linear_system_ode.h"

int main()
{
    const int p = 2; // number of states
    // ode: a_2 * xddot + a_1 * xdot + a_0 * x = b * u;
    const double coeff_a[] {1, -0.1, 1}; // a_2, a_1, a_0
    const double coeff_b = 1; // no derivatives of the input supported atm.

    double x0[] = {0, 0}; // start state
    double xf[] = {1, 0}; // goal state

    // System specific settings
    LinearSystemODE<p> system;
    system.setODECoefficients(coeff_a, coeff_b);

    // Setup controller 1
    teb::SolverLevenbergMarquardtEigenSparse solver1;
    teb::TebController<p,1> teb(&solver1);

    teb.setSystemDynamics(&system);
    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Setup controller 2
    teb::Config mpc_cfg;
    mpc_cfg.optim.solver.lsqr.weight_equalities = 4;
    teb::SolverLevenbergMarquardtEigenSparse solver2;
    teb::TebController<p,1> mpc(&mpc_cfg, &solver2);

    mpc.setSystemDynamics(&system);

    // Setup receding horizon T = n * dt = 10 * 0.1
    // Common setting: unfixed goal and fixed resolution
    mpc.setupHorizon(false, true, 10, 0.1);

    mpc.activateObjectiveQuadraticForm(1, 0.1, 100); // Parameters: Q, R, Qf
    mpc.activateControlBounds(0, -1.0, 1.0);
```

```

// Create simulator
teb::TebPlotter plotter;
teb::Simulator<p,1> sim(system,&plotter);
// Set simulation sample time
sim.setSampleTime(0.05);

//sim.setIntegrator(teb::NumericalIntegrators::EXPLICIT_EULER);

// Create container storing the controllers that should be compared
std::vector<std::pair<teb::BaseController*, std::string>> controllers;
controllers.emplace_back(&teb, "TEB - Time Optimal");
controllers.emplace_back(&mpc, "MPC - Quadratic Form");

// Closed-loop sim
sim.simClosedLoop(controllers, x0, xf, 8);

// Open-loop sim
// create new figure with id 1
plotter.switchWindow(1,true);
sim.simOpenLoop(controllers, x0, xf);

return 0;
}

```

## 18.8 linear\_system\_state\_space.cpp

The following example shows the TEB controller applied to a common linear system described by a continuous-time state space model of the  $p$ -th order:

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \quad \mathbf{x} \in \mathbb{R}^p, \mathbf{u} \in \mathbb{R}^q$$

For this example, let  $p = 2$  and  $q = 1$ . We define  $\mathbf{A} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \end{bmatrix}$  and  $\mathbf{b} = [0, 1]^T$ .

The control input  $u$  should be bounded to the interval  $[-1, 1]$ .

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0]^T$  to the final state  $\mathbf{x}_f = [1, 0]^T$ .

The system dynamics are specified using the LinearSystemStateSpace class. See the following header file how to add the state space model mentioned above.

File **linear\_system\_state\_space.h**:

```

#ifndef __teb_package_examples_linear_system_state_space__
#define __teb_package_examples_linear_system_state_space__

#include <teb_package.h>

template <int p, int q>
class LinearSystemStateSpace : public teb::SystemDynamics<p,q>
{
public:
    using StateVector = typename teb::SystemDynamics<p,q>::StateVector;
    using ControlVector = typename teb::SystemDynamics<p,q>::ControlVector;
    ;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        return _A * x + _B * u;
    }

    void setStateSpaceModel(const Eigen::Ref<const Eigen::Matrix<double,p,p>>& A, const Eigen::Ref<const
        Eigen::Matrix<double,p,q>>& B)
    {
        _A = A;
        _B = B;
    }

protected:
    Eigen::Matrix<double,p,p> _A;
    Eigen::Matrix<double,p,q> _B;
};

#endif /* defined(__teb_package_examples_linear_system_state_space__) */

```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the LinearSystemStateSpace object specified in linear\_system\_state\_space.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:

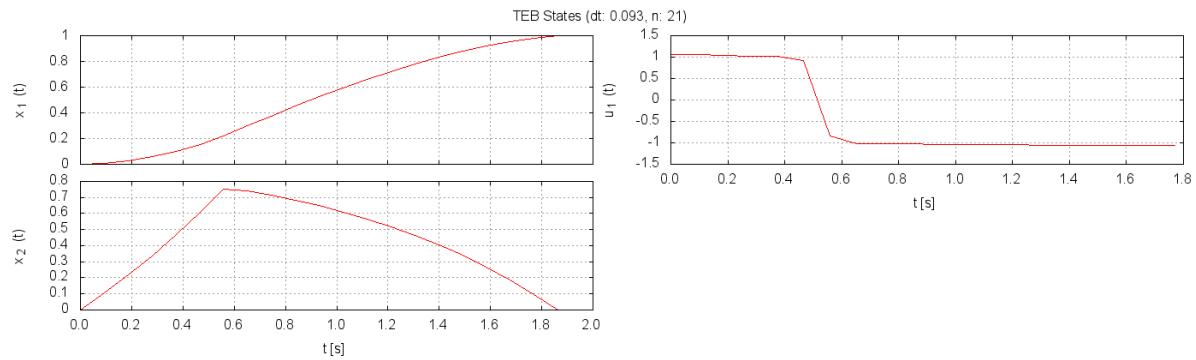


Figure 18.6: Result of linear\_system\_state\_space.cpp

#### File linear\_system\_state\_space.cpp

```
#include "linear_system_state_space.h"

int main()
{
    const int p = 2;
    const int q = 1;

    Eigen::Matrix2d A;
    A(0,0) = 0;
    A(0,1) = 1;
    A(1,0) = 0;
    A(1,1) = 1;

    Eigen::Vector2d b;
    b(0) = 0;
    b(1) = 1;

    double x0[] = {0,0};
    double xf[] = {1,0};
    double u;

    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);

    LinearSystemStateSpace<p,q> system;
    system.setStateSpaceModel(A,b);

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    START_TIMER;
    teb.step(x0, xf, &u);
    STOP_TIMER("TEB optimization loop");

    PRINT_INFO("Control u: " << u);

    teb::TebPlotter plotter;
    plotter.plotTEB(teb);

    return 0;
}
```

## 18.9 mobile\_robot\_teb.cpp

This example shows how to build a customized controller that minimizes a user-defined optimization problem rather than using default optimization functions.

Part of the example is a mobile robot trajectory planning and control application. The optimization problem is given by the original timed-elastic-band presented in [3].

For the sake of simplicity, we ignore some optional cost-functions like constraints on accelerations and the forward drive direction. In addition, we apply only a uniform  $\Delta T$ .

We first start by introducing the optimization problem. The robot pose is defined by  $\mathbf{x} = [x, y, \beta]^T$ . The robot trajectory is then given by the sequence of  $n \in \mathbb{N}$  poses  $\mathbf{x}_i, i = 1, \dots, n$ . Between two consecutive poses  $\{\mathbf{x}_i, \mathbf{x}_{i+1}\}$  we define the time difference  $\Delta T$ , that the robot requires to transit from the a configuration to the subsequent one in the sequence.

The robot hardware interface accepts translational and rotational velocities  $\mathbf{u} = [v, \omega]^T$  as control inputs.

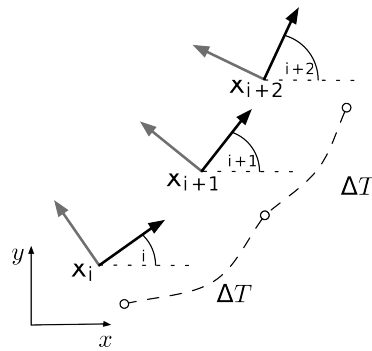


Figure 18.7: Mobile robot trajectory representation

In contrast to common MPC approaches, the sequence of poses and the time difference are part of the optimization rather than the control inputs  $\mathbf{u}$ , because they are implicitly part of the pose sequence and they can be derived via difference quotients.

The joint trajectory representation is then defined as follows:

$$\mathcal{B} := \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{n-1}, \mathbf{x}_n, \Delta T\}$$

In the following we construct the optimization problem.

A general cost function can often be decomposed into the sum of  $z$  "local" cost terms:

$$V(\mathcal{B}) = f(\mathcal{B}) = \sum_i^z f_i(\mathcal{B}_i)$$

$f_i$  denotes a local cost term and  $\mathcal{B}_i \subseteq \mathcal{B}, i = 1, \dots, z$  denotes a (small) subset of  $\mathcal{B}$ .

Obviously, if the subset of depending optimization variables is low for each local cost term, the resulting optimization problem has a sparse structure. To exploit this structure later, a hyper-graph (like mentioned) is ideally suited to capture this property already during the problem formulation phase.

Optimization problems in this package are formulated as hyper-graphs (according to [2]) with additional distinction between constraints and objective functions. A hyper-graph is a graph, in which edges connect an arbitrary number of vertices (not just two). In sense of optimization, edges represent local cost function terms  $f_k(\mathcal{B}_i)$  and vertices represent the optimization variables (in this package commonly  $\mathbf{x}_k, \mathbf{u}_k, \Delta T$ ). Consequently for the mobile robot problem, vertices are:  $\mathbf{x}_k$  and  $\Delta T$  (see the image a few lines below for an example hyper-graph).

The considered mobile robot has two wheels connected with a differential drive. The formulation of the optimization problem in [3] does not take a common robot motion model into account. The kinodynamic motion behavior (constraints on robot / wheel velocities / accelerations and the non-holonomic kinematic) is captured using the superposition of dedicated cost functions for each objective or constraint. For the sake of simplicity we reduce the

optimization task to just limit velocities of the center of the robot, to satisfy non-holonomic kinematics and to avoid obstacles:

$$V(\mathcal{B}) = f(\mathcal{B}) = f_{\text{mintime}}(\Delta T) + \sum_{i=1}^{n-1} [f_{i,\text{vel}}(\mathcal{B}) + f_{i,\text{kin}}(\mathcal{B}) + f_{i,\text{obst}}(\mathcal{B})]$$

Each cost term above is a squared function and hence the optimization problem can be solved using the Levenberg-Marquardt algorithm. Bounding the velocity, satisfying a minimum distance to obstacles and satisfying the kinematic constraints are implemented using soft-constraints in [?].

Within this MPC package, we have the opportunity to specify constraints in a more generalized way. And if the selected solver is a least-squares solver (like [teb::SolverLevenbergMarquardtEigenSparse](#)), then the soft-constraint approach is applied in similar way like in [3]. As a result, we can generalize the above optimization problem for the mobile robot to:

$$V^*(\mathcal{B}) = \min f_{\text{mintime}}(\Delta T) \quad (18.1)$$

s.t.:

$$\mathbf{x}_1 = \mathbf{x}_s \quad (18.2)$$

$$\mathbf{x}_n = \mathbf{x}_f \quad (18.3)$$

$$\mathbf{h}_{i,\text{kin}}(\mathcal{B}) = \mathbf{0} \quad i \in 1, \dots, n-1 \quad (18.4)$$

$$\mathbf{g}_{i,\text{vel}}(\mathcal{B}) \leq \mathbf{0} \quad i \in 1, \dots, n-1 \quad (18.5)$$

$$\mathbf{g}_{i,\text{obst}}(\mathcal{B}) \leq \mathbf{0} \quad i \in 1, \dots, n-1 \quad (18.6)$$

$\mathbf{h}$  and  $\mathbf{g}$  denote the equality and inequality constraints without squared soft-constraint approximations.

According to [3] the cost terms are as follows (**without soft-constraint approx** and **without taking squared**):

- Minimum transition time:  $f_{\text{mintime}}(\Delta T) = (n-1)\Delta T$  (Note, least-square solvers square all objectives later!)
- Limit velocity:

$$\Delta \mathbf{s} = \begin{bmatrix} \left\| \begin{bmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \end{bmatrix} \right\| \\ \text{normAngle}(\beta_{i+1} - \beta_i) \end{bmatrix}$$

$$\mathbf{g}_{i,\text{vel}}(\mathcal{B}) = \frac{1}{\Delta T} \begin{bmatrix} \Delta \mathbf{s} \\ -\Delta \mathbf{s} \end{bmatrix} + \begin{bmatrix} -v_{\text{max}} \\ -\omega_{\text{max}} \\ v_{\text{min}} \\ \omega_{\text{min}} \end{bmatrix}$$

- Nonholonomic Kinematics:

$$\mathbf{d}_i = \begin{bmatrix} x_{i+1} - x_i \\ y_{i+1} - y_i \\ 0 \end{bmatrix}$$

$$\mathbf{h}_{i,\text{kin}} = \left\| \begin{bmatrix} \cos \beta_i \\ \sin \beta_i \\ 0 \end{bmatrix} + \begin{bmatrix} \cos \beta_{i+1} \\ \sin \beta_{i+1} \\ 0 \end{bmatrix} \times \mathbf{d}_i \right\|$$

- Obstacle avoidance (State constraint):

$$\mathbf{g}_{i,\text{obst}} = - \left\| \begin{bmatrix} x_i - x_{\text{obst}} \\ y_i - y_{\text{obst}} \end{bmatrix} \right\| + d_{\text{min}}$$

Please note, that using a least-squares soft-constrained solver (as mentioned above), all constraints and objectives are incorporated in a weighted objective/cost function and are becoming squared. The above optimization problem can easily be transferred to a hyper-graph. See the following figure for the resulting example graph. All indices are increased by one subsequently per constraint/objective type.

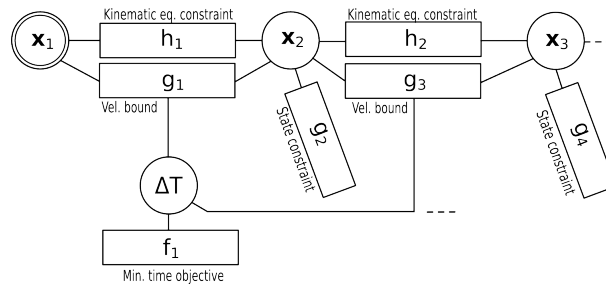


Figure 18.8: Example hyper-graph of the mobile robot application

The first state  $\mathbf{x}_1$  is marked with a double circle. That means the state is fixed during optimization (to define the current/initial state). The goal state is fixed as well (according to a fixed horizon). Fixing these values implicitly satisfies  $\mathbf{x}_1 = \mathbf{x}_s$  and  $\mathbf{x}_n = \mathbf{x}_f$  of the problem definition above.

In contrast to [3] we are using a single  $\Delta T$  that is part of all velocity edges (note the connection of  $\mathbf{g}_1, \mathbf{g}_3$  and  $\Delta T$  in the example graph). Formulating the optimization problem as a hyper-graph immediately indicates the structure of the underlying hessian/jacobian matrix. For example, considering the ordering of the optimization variables of  $\mathcal{B}$ , than the hessian of the Lagrangian or the hessian of a least-square cost-function with soft-constraints would contain a band diagonal (since the maximum number of connected states  $\mathbf{x}_i$  is at least two. Only the velocity bound edges that include always the same  $\Delta T$  would lead to a dense last row and column in the matrix.

The following header-file demonstrates how to implement the "local" cost functions mentioned above (please refer to [teb::BaseEdge](#) for an overview of possible edges to derive from):

File **rob\_cost\_edges.h**:

```
#ifndef __teb_package__examples_rob_cost_edges__
#define __teb_package__examples_rob_cost_edges__

#include <teb_package.h>

namespace teb
{
    // Declarations for cost functions / hyper edges

    // Non-holonomic-Kinematics
    class EdgeKinematics : public BaseEdge<1, FUNCT_TYPE::NONLINEAR, StateVertex<3>, StateVertex<3>> //
    {
        Cost vector dimension: 1, Vertices: StateVertex
    public:
        EdgeKinematics(StateVertex<3>& state1, StateVertex<3>& state2) : BaseEdge<1, FUNCT_TYPE::NONLINEAR,
        StateVertex<3>, StateVertex<3>>(state1, state2) {}

        virtual void computeValues() // See EdgeType::computeValues()
        {
            StateVertex<3>* sample_i = static_cast<StateVertex<3>*>(_vertices[0]);
            StateVertex<3>* sample_ip1 = static_cast<StateVertex<3>*>(_vertices[1]);

            // Direction vector between two samples:
            Eigen::Vector2d deltaS = sample_ip1->states().head(2) - sample_i->states().head(2);
            double angle_i = sample_i->states()[2];
            double angle_ip1 = sample_ip1->states()[2];

            // Nonholonomic Kinematics
            _values[0] = fabs( ( cos(angle_i)+cos(angle_ip1) ) * deltaS[1] - ( sin(angle_i)+sin(angle_ip1)
            ) * deltaS[0] );
        }

        // Hessian and Jacobians are calculated numerically, otherwise add implementation here.

        EIGEN_MAKE_ALIGNED_OPERATOR_NEW
    };

    // Limit Velocity
    class EdgeLimitVelocity : public BaseEdge<4, FUNCT_TYPE::NONLINEAR, StateVertex<3>, StateVertex<3>, Tim
    eDiff> // Cost vector dimension: 4, Vertices: StateVertex, StateVertex, TimeDiff
    {

```

```

public:

    EdgeLimitVelocity(StateVertex<3>& statel, StateVertex<3>& state2, TimeDiff& dt) : BaseEdge<4,
FUNCT_TYPE::NONLINEAR, StateVertex<3>, StateVertex<3>, TimeDiff>(statel, state2, dt)
    {
    }

    virtual void computeValues() // See EdgeType::computeValues()
    {
        StateVertex<3>* sample_i = static_cast<StateVertex<3>*>(_vertices[0]);
        StateVertex<3>* sample_ip1 = static_cast<StateVertex<3>*>(_vertices[1]);
        TimeDiff* dt = static_cast<TimeDiff*>(_vertices[2]);

        // Direction vector between two samples:
        Eigen::Vector2d deltaS = sample_ip1->states().head(2) - sample_i->states().head(2);
        double vel = deltaS.norm() / dt->dt();
        double omega = norm_angle( sample_ip1->states()[2] - sample_i->states()[2] ) / dt->dt

    ();

        // Velocity limits: c(x) < 0
        _values[0] = vel - v_max;
        _values[1] = -vel + v_min;
        _values[2] = omega - omega_max;
        _values[3] = -omega + omega_min;
    }

    // Hessian and Jacobians are calculated numerically, otherwise add implementation here.

    double v_max = 1;
    double v_min = -1;
    double omega_max = 0.5;
    double omega_min = -0.5;

    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

// Obstacle-Avoidance (point-representation)
class EdgeObstacle : public BaseEdge<1, FUNCT_TYPE::NONLINEAR, StateVertex<3>> // Cost vector
    dimension: 1, Vertices: StateVertex
{
public:

    EdgeObstacle(StateVertex<3>& state) : BaseEdge<1, FUNCT_TYPE::NONLINEAR, StateVertex<3>>(state) {}

    virtual void computeValues() // See EdgeType::computeValues()
    {
        StateVertex<3>* sample_i = static_cast<StateVertex<3>*>(_vertices[0]);

        // Direction vector between sample and obstacle:
        Eigen::Vector2d deltaS = sample_i->states().head(2) - obstacle_position;
        //double angle_i = sample_i->states()[2];

        // Projection of the distance deltaS orthogonal to the TEB
        //double angdiff = atan2(deltaS[1],deltaS[0])-angle_i;
        double proj_dist = deltaS.norm(); //fabs(sin(angdiff));

        // Constraint proj_dist > min_distance -> -proj_dist + min_distance < 0
        _values[0] = -proj_dist + min_distance;
    }

    // Hessian and Jacobians are calculated numerically, otherwise add implementation here.

    Eigen::Vector2d obstacle_position; // DO NOT FORGET TO SET VALUE
    double min_distance = 0.5;

    EIGEN_MAKE_ALIGNED_OPERATOR_NEW
};

} // end namespace teb

#endif /* defined(__teb_package__examples_rob_cost_edges__) */

```

In order to add the hyper-graph to the `TebController`, you can derive the class `teb::TebController` an override the method `teb::TebController::customOptimizationGraph()`. This ensures that the default edges like `teb::EdgeMinimizeTime` or `teb::EdgeQuadraticForm` can be added as well using the controller API. You can change states to fixed or unfixed as well. (Please note, that the default fixing and unfixing of the start and goal state as well as the default edges for the hyper-graph are applied in `teb::TebController::buildOptimizationGraph()`).



The derived class for the mobile-robot example is defined as follows. (**Note**, since the problem is defined with only implicit control inputs, by means of  $q=0$ , the `teb::TebController::firstControl()` and `teb::TebController::returnControllInputSequence()` are overridden as well. Additionally, we extend the new controller API with more debug and visualization functions.

File `rob_controller.h`:

```
#ifndef __teb_package__examples_rob_controller__
#define __teb_package__examples_rob_controller__

#include <teb_package.h>
#include "rob_cost_edges.h"

namespace teb
{
    // Controller implementation
    class RobController : public TebController<3,0>
    {
    public:

#ifdef WIN32 // VS2013 does not support constructor inheritance (only starting from Nov 2013 CTP)
        RobController(BaseSolver* solver = nullptr) : TebController(solver) {};
        RobController(const Config* config, BaseSolver* solver = nullptr) : TebController(config, solver) {
};
#else
        using TebController<3,0>::TebController; // inherit base constructors
#endif

        // Overload customOptimizationGraph() to define a custom hyper-graph
        // and thus preserve the option to add predefined edges like time-optimality
        virtual void customOptimizationGraph()
        {
            for (unsigned int i=0; i<getN()-1; ++i)
            {
                // Add non-holonomic kinematics
                EdgeKinematics* kinematics_edge = new EdgeKinematics(_state_seq.at(i), _state_seq.at(i + 1));
            };
            _graph.addEdgeEquality(kinematics_edge);

            // Add velocity limits (translational and rotational)
            EdgeLimitVelocity* vel_edge = new EdgeLimitVelocity(_state_seq.at(i), _state_seq.at(i + 1),
            _dt);
            _graph.addEdgeInequality(vel_edge);

            // Add single obstacle edge
            EdgeObstacle* obst_edge = new EdgeObstacle(_state_seq.at(i));
            obst_edge->min_distance = 0.5;
            obst_edge->obstacle_position = _obstacle_pos;
            _graph.addEdgeInequality(obst_edge);
        }

};

void setObstaclePosition(const Eigen::Ref<const Eigen::Vector2d>& obstacle_pos) {_obstacle_pos =
obstacle_pos;};
void setObstaclePosition(const double* obstacle_pos) {_obstacle_pos = Eigen::Vector2d(obstacle_pos);};

void setCurrentVelocity(const Eigen::Ref<const Eigen::Vector2d>& curr_velocity) {_curr_velocity =
curr_velocity;};
void setCurrentVelocity(double v, double omega) {_curr_velocity[0]=v; _curr_velocity[1]=omega;};

// Get all velocities (implicit from states)
// This method is not efficient, but it is only used for visualization here.
// Returns a [2 x n-1] matrix in which each column denotes: [sqrt(vx^2+vy^2), omega]^T
Eigen::MatrixXd getVelocities()
{
    unsigned int n = getN(); // Number of states
    Eigen::MatrixXd teb_mat = getStateCtrlInfoMat(); // inefficient here, but comfortable
    Eigen::Matrix<double,3,-1> vel3 = (teb_mat.topRightCorner(3,n-1) - teb_mat.topLeftCorner(3,n-1));
    // Normalize angle difference
    norm_angle_vec(vel3.bottomRows(0));
    vel3 /= dt().dt(); // scale to actual velocities

    Eigen::MatrixXd vel2(2,vel3.cols());
    vel2.row(0) = vel3.topRows(2).colwise().norm();
    vel2.row(1) = vel3.row(2);
    return vel2;
}
```

```

// Override original function to return the current control input, since
// we use an implicit control input model (as a function of the states)
Eigen::VectorXd firstControl() const
{
    Eigen::VectorXd u(2,1);
    u[0] = (_state_seq.at(1).states().head(2) - _state_seq.at(0).states().head(2)).norm() / dt().dt();
    // transl. velocity
    u[1] = norm_angle((_state_seq.at(1).states()[2] - _state_seq.at(0).states()[2])) / dt().dt(); // rot. velocity

// Check if Goal reached:
if ( (firstStateRef() - lastStateRef()).norm() < 0.1)
{
    u.setZero();
    PRINT_INFO_ONCE("RobController: Goal Reached.");
}

    return u;
}

// Override this function as well to get all implicit controls at once
virtual Eigen::MatrixXd returnControlInputSequence() const
{
    Eigen::MatrixXd states = getStateCtrlInfoMat();

    unsigned int m = getN()-1;
    Eigen::MatrixXd u_seq(2,m);

    u_seq.row(0) = ( states.topRightCorner(2,m) - states.topLeftCorner(2,m) ).colwise().norm() / getDt();
    u_seq.row(1) = states.bottomRightCorner(1, m) - states.bottomLeftCorner(1, m);
    norm_angle_vec(u_seq.bottomRows(0));
    u_seq.row(1) /= getDt();

    return u_seq;
}

// Update current velocity after each step to initialize the subsequent step
void robSaveVelocityAfterStep(BaseController* ctrl, SystemDynamics<3,2>* system, Simulator<3,2>* sim)
{
    setCurrentVelocity(firstControl());
}

// We need some additional visualization
void plotRobotStuff()
{
    // Plot states and control input
    TebPlotter plotter;
    plotter.plotTEB(*this);

    // Create x-y plot
    plotter.switchWindow(1,true); // Create new window
    Eigen::MatrixXd teb_mat = getStateCtrlInfoMat(); // Get all TEB states in a single matrix
    plotter.plot(teb_mat.row(0).transpose(), teb_mat.row(1).transpose(),"X-Y Trajectory","x [m]", "y [m]");

    // Get velocity from the states
    Eigen::MatrixXd vel = getVelocities();

    // plot resulting translational velocity
    plotter.switchWindow(2,true);

    PlotOptions options;
    options.title = "Robot velocities";
    options.ylabels.emplace_back("trans. vel [m/s]");
    options.ylabels.emplace_back("rot. vel [rad/s]");

    plotter.plotMulti(getAbsoluteTimeVec().head(getN()-1), vel, &options);
}

protected:
    Eigen::Vector2d _curr_velocity = Eigen::Vector2d::Zero(); // [v, omega]^T
    Eigen::Vector2d _obstacle_pos = Eigen::Vector2d::Zero();
};

// Velocity bounds

```

```

} // end namespace teb

#endif /* defined(__teb_package__examples_rob_controller__) */

```

Furthermore, for simulation purposes we define a `teb::SystemDynamics` model with simple integrator dynamics and the non-holonomic kinematics:

File `mobile_robot_teb.h`:

```

#ifndef __teb_package__examples_mobile_robot_teb__
#define __teb_package__examples_mobile_robot_teb__

#include <teb_package.h>

#include "rob_controller.h" // include customized controller

// Specify system model (here for simulation only, but could be used for the controller as well (as an
// alternative))
// We use a simple kinematic model here with integrator dynamics
class MobileRobot : public teb::SystemDynamics<3,2>
{
public:
    using StateVector = teb::SystemDynamics<3,2>::StateVector;
    using ControlVector = teb::SystemDynamics<3,2>::ControlVector;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        // kinematic motion constraints
        // states = [x y theta]
        // inputs = [v, omega]

        state_equations[0] = u[0] * std::cos(x[2]); // xdot = v*cos(theta)
        state_equations[1] = u[0] * std::sin(x[2]); // ydot = v*sin(theta)
        state_equations[2] = norm_angle(u[1]); // thetadot = omega
        return state_equations;
    }
};

#endif /* defined(__teb_package__examples_mobile_robot_teb__) */

```

Finally, the main function that creates all objects, defines the optimization weights and initializes the simulation can be found in `mobile_robot_teb.cpp` (see below at the end).

The output of the program is as follows (the **obstacle** is placed at  $x_{obst} = 2.5$  m and  $y_{obst} = 0.1$  m):

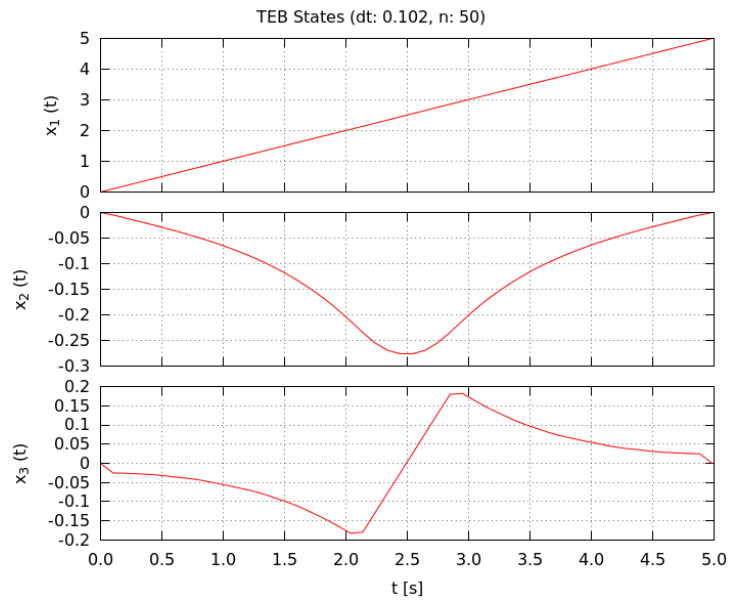


Figure 18.9: TEB States after the first open-loop optimization

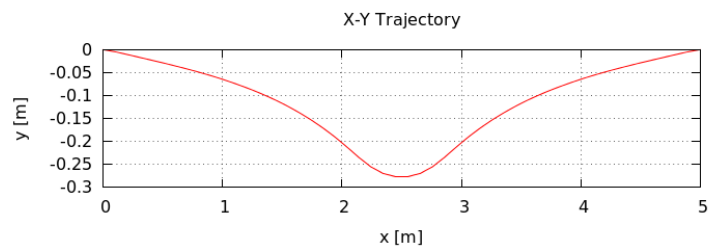


Figure 18.10: X-Y Trajectory after the first open-loop optimization

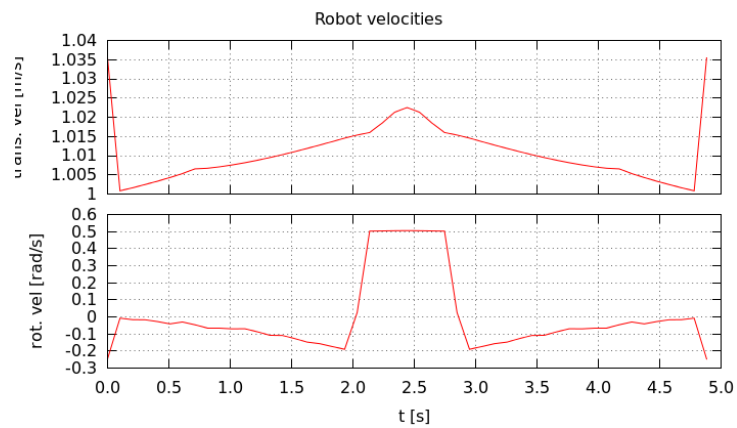


Figure 18.11: Velocity profile after the first open-loop optimization

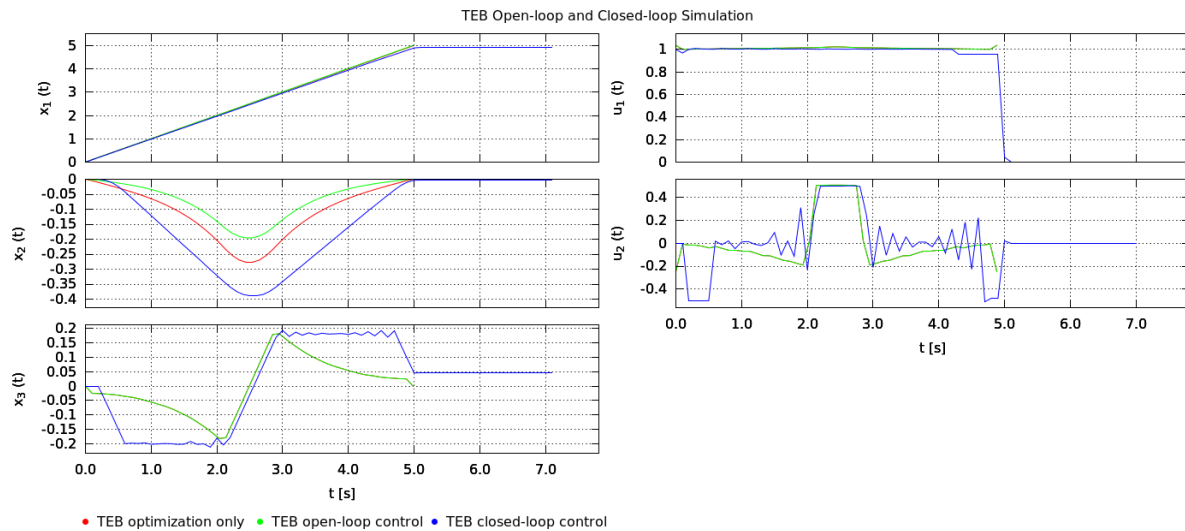


Figure 18.12: Closed-loop simulation of the mobile robot

File **mobile\_robot\_teb.cpp**:

```
#include "mobile_robot_teb.h"

int main()
{
    teb::Config cfg;
    cfg.optim.solver.lsqr.weight_equalities = 500;
    cfg.optim.solver.lsqr.weight_inequalities = 50;
    cfg.optim.solver.lsqr.weight_adaptation_factor = 2;
    cfg.teb.dt_min = 0.05;
    cfg.teb.n_pre = 50;
    cfg.teb.n_max = 150;
    cfg.teb.n_min = 2;

    double x0[] = {0, 0, 0};
    double xf[] = {5, 0, 0};
    double obst_pos[] = {2.5, 0.1};

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::RobController teb(&cfg, &solver);

    teb.activateObjectiveTimeOptimal();
    teb.setObstaclePosition(obst_pos);

    // Perform open-loop planning
    START_TIMER;
    teb.step(x0, xf);
    STOP_TIMER("TEB optimization loop");

    // Plot open-loop results
    teb.plotRobotStuff();

    // Perform closed-loop sim
    teb::TebPlotter plotter;
    MobileRobot robot; // Motion model for simulation
    teb::Simulator<3,2> sim(&teb, robot, &plotter);

    // Set simulation sample time
    sim.setSampleTime(0.1);

    using namespace std::placeholders;
    sim.setPostStepCallback(std::bind(&teb::RobController::robSaveVelocityAfterStep, &
        teb, _1, _2, _3));

    // Start simulation
    sim.simOpenAndClosedLoop(x0, xf, 7);

    return 0;
}
```

## 18.10 rocket\_system.cpp

The following example shows the TEB controller applied to a simple free space rocket model with three differential states  $s$ ,  $v$  and  $m$ . ([http://acado.sourceforge.net/doc/html/d9/d65/example\\_001.html](http://acado.sourceforge.net/doc/html/d9/d65/example_001.html))

$$\dot{s}(t) = v(t) \quad (18.7)$$

$$\dot{v}(t) = \frac{u(t) - 0.02v^2(t)}{m(t)} \quad (18.8)$$

$$\dot{m}(t) = -0.01u^2(t) \quad (18.9)$$

The control input  $u$  should be bounded to the interval  $[-1.1, 1.1]$ . In addition the state  $v$  should be bounded to the interval  $[-0.5, 1.7]$ .

The state vector is defined by  $\mathbf{x} = [s, v, m]^T$ .

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0, 1]^T$  to the final state  $\mathbf{x}_f = [10, 0, -]^T$ .

In addition to the other examples, the third state of the final state vector is not fixed. The final mass is not known a-priori and can be chosen by the optimizer in order to minimize the objective function. But we initialize/estimate the final value to  $m_f = 0.5$ .

The system dynamics are specified using the FreeSpaceRocketSystem class. See the following header file how to add the nonlinear state space model mentioned above.

File **rocket\_system.h**:

```
#ifndef __teb_package__examples_rocket_system__
#define __teb_package__examples_rocket_system__

#include <teb_package.h>

class FreeSpaceRocketSystem : public teb::SystemDynamics<3,1>
{
public:
    using StateVector = teb::SystemDynamics<3,1>::StateVector;
    using ControlVector = teb::SystemDynamics<3,1>::ControlVector;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        state_equations[0] = x[1]; // sdot = v
        state_equations[1] = (u[0] - 0.02 * x[1] * x[1]) / x[2]; // (u-0.02*v^2)/m
        state_equations[2] = -0.01 * u[0] * u[0]; // -0.01 * u^2

        return state_equations;
    }
};

#endif /* defined(__teb_package__examples_rocket_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the FreeSpaceRocketSystem object specified in van\_der\_pol\_system.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows (**Note**, the number of iterations and weights of the soft constraints is too low using the default settings, therefore the constraints are not satisfied completely after the first optimization):

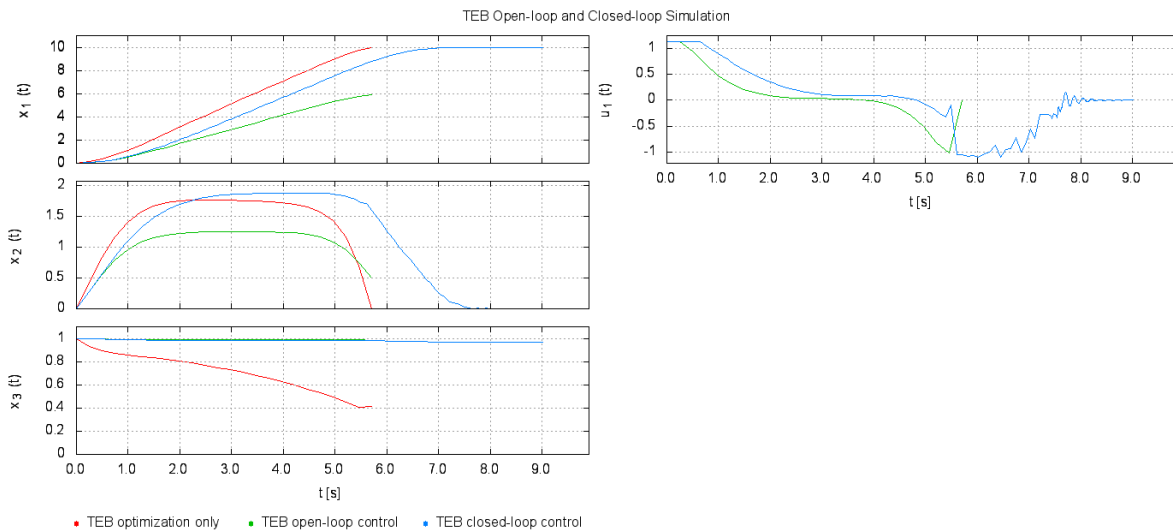


Figure 18.13: Result of rocket\_system.cpp

## File rocket\_system.cpp

```
#include "rocket_system.h"

int main()
{
    const int p = 3; // number of states

    // Change default config, since default soft constraint weights are too low.
    teb::Config cfg;
    cfg.optim.solver.lsqr.weight_inequalities = 50;
    cfg.optim.solver.lsqr.weight_adaptation_factor = 1.1;
    cfg.optim.solver.lsqr.soft_constr_epsilon = 0;

    // Define start and goal
    double x0[] = {0,0,1}; // start state
    double xf[] = {10,0,0.5}; // goal state
    bool goal_fixed[] = {true,true,false}; // mass is not fixed

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&cfg, &solver);

    // System specific settings
    FreeSpaceRocketSystem system;

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.1, 1.1);
    teb.activateStateBounds(1,-0.5,1.7);
    teb.setGoalStatesFixedOrUnfixed(goal_fixed);

    // Create simulator
    teb::TebPlotter plotter;
    teb::Simulator<p,1> sim(&teb,system,&plotter);

    // Set simulation sample time
    sim.setSampleTime(-1); // Inherit from TEB (asynchronous control)

    // Simulate 9 seconds
    sim.simOpenAndClosedLoop(x0,xf,9);

    return 0;
}
```

## 18.11 van\_der\_pol\_system.cpp

The following example shows the TEB controller applied to the Van der Pol oscillator ([http://en.wikipedia.org/wiki/Van\\_der\\_Pol\\_oscillator](http://en.wikipedia.org/wiki/Van_der_Pol_oscillator))

$$\ddot{x}(t) - a(1 - x^2(t))\dot{x}(t) + x(t) = u(t)$$

The control input  $u$  should be bounded to the interval  $[-1, 1]$ .

The state vector is defined by  $\mathbf{x} = [x, \dot{x}]^T$ . In this example it is  $a = 1$ .

The objective of the control task is to transite the system from the start state  $\mathbf{x}_s = [0, 0]^T$  to the final state  $\mathbf{x}_f = [1, 0]^T$ .

The system dynamics are specified using the VanDerPolSystem class. See the following header file how to add the nonlinear state space model mentioned above.

File **van\_der\_pol\_system.h**:

```
#ifndef __teb_package__examples_van_der_pol_system__
#define __teb_package__examples_van_der_pol_system__

#include <teb_package.h>

class VanDerPolSystem : public teb::SystemDynamics<2,1>
{
public:
    using StateVector = teb::SystemDynamics<2,1>::StateVector;
    using ControlVector = teb::SystemDynamics<2,1>::ControlVector;

    inline virtual StateVector stateSpaceModel(const Eigen::Ref<const StateVector>& x, const
        Eigen::Ref<const ControlVector>& u) const
    {
        StateVector state_equations;

        state_equations[0] = x[1]; // xdot = xdot
        state_equations[1] = -_a * (x[0] - 1) * x[1] - x[0] + u[0];

        return state_equations;
    }

    void setParameters(double a) {_a = a;};

protected:
    double _a = 1;
};

#endif /* defined(__teb_package__examples_van_der_pol_system__) */
```

The following cpp-file shows how to create the TebController object, how to add control bounds and how to add the VanDerPolSystem object specified in van\_der\_pol\_system.h.

The example program solves the optimization problem once without actually controlling the system. See other examples for open-loop and closed-loop control applications.

The output of the program is as follows:



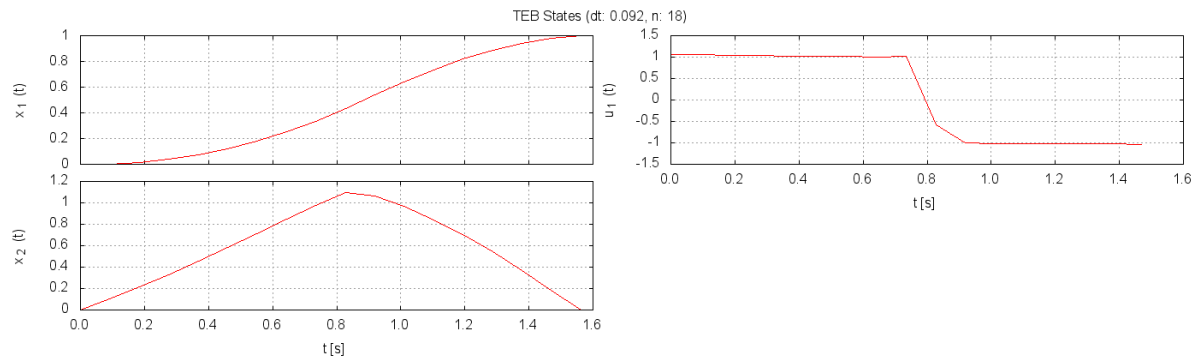


Figure 18.14: Result of van\_der\_pol\_system.cpp

## File van\_der\_pol\_system.cpp

```
#include "van_der_pol_system.h"

int main()
{
    const int p = 2; // number of states

    // Define start and goal
    double x0[] = {0,0}; // start state
    double xf[] = {1,0}; // goal state
    double u;

    // Setup solver and controller
    teb::SolverLevenbergMarquardtEigenSparse solver;
    teb::TebController<p,1> teb(&solver);

    // System specific settings
    VanDerPolSystem system;
    system.setParameters(1.0);

    teb.setSystemDynamics(&system);

    teb.activateObjectiveTimeOptimal();
    teb.activateControlBounds(0, -1.0, 1.0);

    // Perform open-loop planning
    START_TIMER;
    teb.step(x0, xf, &u);
    STOP_TIMER("TEB optimization loop");

    // Print determined control for the current sampling interval
    PRINT_INFO("Control u: " << u);

    // Plot states and control input
    teb::TebPlotter plotter;
    plotter.plotTEB(teb);

    return 0;
}
```



# Bibliography

- [1] Martin Keller, Frank Hoffmann, Torsten Bertram, Carsten Hass, and Alois Seewald. Planning of optimal collision avoidance trajectories with timed elastic bands. In *19th World Congress of the International Federation of Automatic Control*, pages 9822–9827, 2014. [2](#)
- [2] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. In *IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. [2](#), [9](#), [53](#), [63](#), [80](#), [149](#), [204](#), [207](#), [216](#), [219](#), [359](#)
- [3] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Trajectory modification considering dynamic constraints of autonomous robots. In *7th German Conference on Robotics (ROBOTIK)*, pages 74–79, Berlin and Offenbach, 2012. VDE-Verl. [2](#), [359](#), [360](#), [361](#)
- [4] Christoph Rösmann, Wendelin Feiten, Thomas Wösch, Frank Hoffmann, and Torsten Bertram. Efficient trajectory optimization using a sparse model. In *European Conference on Mobile Robots*, pages 138–143, 2013. [2](#)
- [5] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Prädiktive regelung mit timed-elastic-bands. *at - Automatisierungstechnik*, 62(10):720–731, 2014. [1](#), [293](#)

# Index

- ~BaseEdge
  - teb::BaseEdge, [52](#)
  - teb::BaseEdge< D, FuncType >, [61](#)
- ~BaseSolver
  - teb::BaseSolver, [69](#)
- ~EdgeType
  - teb::EdgeType, [155](#)
- ~ExplicitEuler
  - teb::ExplicitEuler, [160](#)
- ~HessianWorkspace
  - teb::HessianWorkspace, [164](#)
- ~JacobianWorkspace
  - teb::JacobianWorkspace, [173](#)
- ~NumericalIntegrator
  - teb::NumericalIntegrator, [180](#)
- ~RungeKutta5thOrder
  - teb::RungeKutta5thOrder, [184](#)
- ~RungeKuttaClassic
  - teb::RungeKuttaClassic, [186](#)
- ~Simulator
  - teb::Simulator, [193](#)
- ~SystemDynamics
  - teb::SystemDynamics, [267](#)
- ~TebController
  - teb::TebController, [277](#)
- ~TebPlotter
  - teb::TebPlotter, [302](#)
- ~TimeDiff
  - teb::TimeDiff, [311](#)
- \_A
  - teb::SolverSQPDense, [250](#)
- \_Q
  - teb::EdgeQuadraticForm, [143](#)
- \_R
  - teb::EdgeQuadraticForm, [143](#)
- \_active\_control\_bounds
  - teb::TebController, [296](#)
- \_active\_quadratic\_form
  - teb::TebController, [297](#)
- \_active\_state\_bounds
  - teb::TebController, [297](#)
- \_active\_system\_dynamics
  - teb::TebController, [297](#)
- \_active\_time\_optimal
  - teb::TebController, [297](#)
- \_active\_vertices
  - teb::BaseSolver, [73](#)
  - teb::BaseSolverLeastSquares, [84](#)
  - teb::BaseSolverNonlinearProgramDense, [97](#)
  - teb::HyperGraph, [171](#)
  - teb::SolverLevenbergMarquardtEigenDense, [210](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [223](#)
  - teb::SolverNloptPackage, [234](#)
  - teb::SolverSQPDense, [250](#)
- \_backup
  - teb::ControlVertex, [123](#)
  - teb::StateVertex, [265](#)
  - teb::TimeDiff, [314](#)
- \_bound\_constraints
  - teb::SolverNloptPackage, [234](#)
- \_bound\_constraints\_dim
  - teb::SolverNloptPackage, [234](#)
- \_bounds
  - teb::BoundConstraint, [110](#)
- \_callback\_sim\_pre
  - teb::Simulator, [199](#)
- \_callback\_step\_post
  - teb::Simulator, [199](#)
- \_callback\_step\_pre
  - teb::Simulator, [199](#)
- \_cfg\_owned
  - teb::TebController, [298](#)
- \_constraints\_eq
  - teb::HyperGraph, [171](#)
- \_constraints\_ineq
  - teb::HyperGraph, [171](#)
- \_control\_bounds
  - teb::Simulator, [199](#)
- \_controller
  - teb::Simulator, [199](#)
- \_controls
  - teb::ControlVertex, [124](#)
- \_ctol\_eq
  - teb::SolverNloptPackage, [234](#)
- \_ctol\_ineq
  - teb::SolverNloptPackage, [234](#)
- \_ctrl\_seq
  - teb::TebController, [298](#)
- \_data
  - teb::EdgeMinimizeTime, [133](#)
- \_delta
  - teb::SolverSQPDense, [250](#)
- \_dimension
  - teb::ControlVertex, [124](#)
  - teb::StateVertex, [265](#)
- \_dmultiplier\_eq
  - teb::SolverSQPDense, [250](#)
- \_dmultiplier\_ineq

- teb::SolverSQPDense, 250
- \_dt
  - teb::TebController, 298
  - teb::TimeDiff, 314
- \_dt\_ref
  - teb::TebController, 298
- \_equalities
  - teb::BaseSolver, 73
  - teb::BaseSolverLeastSquares, 84
  - teb::BaseSolverNonlinearProgramDense, 97
  - teb::SolverLevenbergMarquardtEigenDense, 210
  - teb::SolverLevenbergMarquardtEigenSparse, 223
  - teb::SolverNloptPackage, 234
  - teb::SolverSQPDense, 251
- \_equalities\_dim
  - teb::BaseSolver, 73
  - teb::BaseSolverLeastSquares, 84
  - teb::BaseSolverNonlinearProgramDense, 97
  - teb::SolverLevenbergMarquardtEigenDense, 211
  - teb::SolverLevenbergMarquardtEigenSparse, 223
  - teb::SolverNloptPackage, 234
  - teb::SolverSQPDense, 251
- \_equality\_jacobian
  - teb::BaseSolverNonlinearProgramDense, 97
  - teb::SolverSQPDense, 251
- \_equality\_values
  - teb::BaseSolverNonlinearProgramDense, 97
  - teb::SolverSQPDense, 251
- \_file\_export
  - teb::TebPlotter, 307
- \_fixed
  - teb::TimeDiff, 315
- \_fixed\_ctrls
  - teb::ControlVertex, 124
- \_fixed\_goal\_states
  - teb::TebController, 298
- \_fixed\_states
  - teb::StateVertex, 265
- \_goal\_backup
  - teb::TebController, 298
- \_graph
  - teb::TebController, 298
- \_graph\_modified
  - teb::HyperGraph, 171
- \_graph\_structure\_modified
  - teb::BaseSolver, 73
  - teb::BaseSolverLeastSquares, 84
  - teb::BaseSolverNonlinearProgramDense, 97
  - teb::SolverLevenbergMarquardtEigenDense, 211
  - teb::SolverLevenbergMarquardtEigenSparse, 223
  - teb::SolverNloptPackage, 235
  - teb::SolverSQPDense, 251
- \_hessians
  - teb::BaseEdge, 56
  - teb::BaseEdge< D, FuncType >, 66
  - teb::BoundConstraint, 110
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 142
  - teb::EdgeSystemDynamics, 152
  - teb::EdgeType, 158
- \_increment
  - teb::SolverSQPDense, 251
- \_inequalities
  - teb::BaseSolver, 74
  - teb::BaseSolverLeastSquares, 84
  - teb::BaseSolverNonlinearProgramDense, 98
  - teb::SolverLevenbergMarquardtEigenDense, 211
  - teb::SolverLevenbergMarquardtEigenSparse, 224
  - teb::SolverNloptPackage, 235
  - teb::SolverSQPDense, 252
- \_inequalities\_dim
  - teb::BaseSolver, 74
  - teb::BaseSolverLeastSquares, 85
  - teb::BaseSolverNonlinearProgramDense, 98
  - teb::SolverLevenbergMarquardtEigenDense, 211
  - teb::SolverLevenbergMarquardtEigenSparse, 224
  - teb::SolverNloptPackage, 235
  - teb::SolverSQPDense, 252
- \_inequalities\_without\_bounds
  - teb::SolverNloptPackage, 235
- \_inequalities\_without\_bounds\_dim
  - teb::SolverNloptPackage, 235
- \_inequality\_jacobian
  - teb::BaseSolverNonlinearProgramDense, 98
  - teb::SolverSQPDense, 252
- \_inequality\_values
  - teb::BaseSolverNonlinearProgramDense, 98
  - teb::SolverSQPDense, 252
- \_integrator
  - teb::Simulator, 199
- \_jacob\_backup
  - teb::BaseEdge, 56
  - teb::BaseEdge< D, FuncType >, 66
  - teb::BoundConstraint, 110
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 142
  - teb::EdgeSystemDynamics, 152
  - teb::EdgeType, 158
- \_jacobian
  - teb::SolverLevenbergMarquardtEigenDense, 211
  - teb::SolverLevenbergMarquardtEigenSparse, 224
- \_jacobians
  - teb::BaseEdge, 57
  - teb::BaseEdge< D, FuncType >, 66
  - teb::BoundConstraint, 111
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 142
  - teb::EdgeSystemDynamics, 152
  - teb::EdgeType, 159
- \_lagrangian\_gradient
  - teb::BaseSolverNonlinearProgramDense, 98
  - teb::SolverSQPDense, 252
- \_lagrangian\_gradient\_backup
  - teb::BaseSolverNonlinearProgramDense, 98
  - teb::SolverSQPDense, 252
- \_lagrangian\_hessian
  - teb::BaseEdge, 56
  - teb::BaseEdge< D, FuncType >, 66
  - teb::BoundConstraint, 110
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 142
  - teb::EdgeSystemDynamics, 152
  - teb::EdgeType, 158

- teb::BaseSolverNonlinearProgramDense, 98
- teb::SolverSQPDense, 253
- \_lb
  - teb::SolverSQPDense, 253
- \_lower\_bounds
  - teb::SolverNloptPackage, 235
- \_merit\_alpha
  - teb::SolverSQPDense, 253
- \_merit\_grad
  - teb::SolverSQPDense, 253
- \_multiplier\_eq
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::SolverSQPDense, 253
- \_multiplier\_ineq
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::SolverSQPDense, 253
- \_nnz\_per\_col
  - teb::SolverLevenbergMarquardtEigenSparse, 224
- \_no\_samples
  - teb::TebController, 299
- \_no\_vert\_backups
  - teb::BaseSolver, 74
  - teb::BaseSolverLeastSquares, 85
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::SolverLevenbergMarquardtEigenDense, 211
  - teb::SolverLevenbergMarquardtEigenSparse, 224
  - teb::SolverNloptPackage, 235
  - teb::SolverSQPDense, 253
- \_objective\_dim
  - teb::BaseSolver, 74
  - teb::BaseSolverLeastSquares, 85
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::SolverLevenbergMarquardtEigenDense, 212
  - teb::SolverLevenbergMarquardtEigenSparse, 224
  - teb::SolverNloptPackage, 236
  - teb::SolverSQPDense, 253
- \_objective\_gradient
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::SolverSQPDense, 254
- \_objective\_value
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::SolverSQPDense, 254
- \_objectives
  - teb::BaseSolver, 74
  - teb::BaseSolverLeastSquares, 85
  - teb::BaseSolverNonlinearProgramDense, 99
  - teb::HyperGraph, 171
  - teb::SolverLevenbergMarquardtEigenDense, 212
  - teb::SolverLevenbergMarquardtEigenSparse, 224
  - teb::SolverNloptPackage, 236
  - teb::SolverSQPDense, 254
- \_opt\_vec\_dim
  - teb::BaseSolver, 74
  - teb::BaseSolverLeastSquares, 85
  - teb::BaseSolverNonlinearProgramDense, 100
  - teb::SolverLevenbergMarquardtEigenDense, 212
  - teb::SolverLevenbergMarquardtEigenSparse, 225
  - teb::SolverNloptPackage, 236
- teb::SolverSQPDense, 254
- \_opt\_vec\_idx
  - teb::ControlVertex, 124
  - teb::StateVertex, 265
  - teb::TimeDiff, 315
  - teb::VertexType, 321
- \_optimized
  - teb::TebController, 299
- \_pdf\_flag
  - teb::TebPlotter, 307
- \_plotter
  - teb::Simulator, 200
- \_qdual
  - teb::SolverSQPDense, 254
- \_qsolver
  - teb::SolverSQPDense, 254
- \_ref\_state
  - teb::EdgeQuadraticForm, 143
- \_sample\_time
  - teb::Simulator, 200
- \_solver
  - teb::TebController, 299
- \_sparse\_solver
  - teb::SolverLevenbergMarquardtEigenSparse, 225
- \_state\_seq
  - teb::TebController, 299
- \_states
  - teb::StateVertex, 265
- \_system
  - teb::EdgeSystemDynamics, 152
  - teb::Simulator, 200
- \_ub
  - teb::SolverSQPDense, 254
- \_upper\_bounds
  - teb::SolverNloptPackage, 236
- \_val\_dim
  - teb::BaseSolverLeastSquares, 85
  - teb::SolverLevenbergMarquardtEigenDense, 212
  - teb::SolverLevenbergMarquardtEigenSparse, 225
- \_values
  - teb::BaseEdge, 57
  - teb::BaseEdge< D, FuncType >, 67
  - teb::BaseSolverLeastSquares, 86
  - teb::BoundConstraint, 111
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 143
  - teb::EdgeSystemDynamics, 152
  - teb::SolverLevenbergMarquardtEigenDense, 212
  - teb::SolverLevenbergMarquardtEigenSparse, 225
- \_vertices
  - teb::BaseEdge, 57
  - teb::BaseEdge< D, FuncType >, 67
  - teb::BoundConstraint, 111
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 143
  - teb::EdgeSystemDynamics, 152
- \_vertices\_dim
  - teb::BaseEdge, 57

- teb::BaseEdge< D, FuncType >, 67
- teb::BoundConstraint, 111
- teb::EdgeMinimizeTime, 133
- teb::EdgeQuadraticForm, 143
- teb::EdgeSystemDynamics, 153
- \_weight\_adapt\_count
  - teb::BaseSolverLeastSquares, 86
  - teb::SolverLevenbergMarquardtEigenDense, 212
  - teb::SolverLevenbergMarquardtEigenSparse, 225
- \_weight\_equalities
  - teb::BaseSolverLeastSquares, 86
  - teb::SolverLevenbergMarquardtEigenDense, 213
  - teb::SolverLevenbergMarquardtEigenSparse, 225
- \_weight\_inequalities
  - teb::BaseSolverLeastSquares, 86
  - teb::SolverLevenbergMarquardtEigenDense, 213
  - teb::SolverLevenbergMarquardtEigenSparse, 225
- \_workspace
  - teb::HessianWorkspace, 165
  - teb::JacobianWorkspace, 174
- ALL
  - teb, 42
- activateControlBounds
  - teb::TebController, 277, 278
- activateObjectiveQuadraticForm
  - teb::TebController, 278
- activateObjectiveTimeOptimal
  - teb::TebController, 279
- activateStateBounds
  - teb::TebController, 279, 280
- activeVertices
  - teb::HyperGraph, 167
- adaptWeights
  - teb::BaseSolverLeastSquares, 78
  - teb::SolverLevenbergMarquardtEigenDense, 204
  - teb::SolverLevenbergMarquardtEigenSparse, 217
- addActiveVertex
  - teb::HyperGraph, 167
- addEdgeEquality
  - teb::HyperGraph, 167
- addEdgeInequality
  - teb::HyperGraph, 167
- addEdgeObjective
  - teb::HyperGraph, 169
- algorithm
  - teb::Config::Optim::Solver::Nlopt, 177
- allocate
  - teb::HessianWorkspace, 164
  - teb::JacobianWorkspace, 173
- allocateMemory
  - teb::BaseEdge, 52
  - teb::BaseEdge< D, FuncType >, 61
  - teb::BoundConstraint, 104
  - teb::EdgeMinimizeTime, 129
  - teb::EdgeQuadraticForm, 138
  - teb::EdgeSystemDynamics, 147
  - teb::EdgeType, 156
  - teb::SimResults, 188
- allocateSparseJacobian
  - teb::SolverLevenbergMarquardtEigenSparse, 217
- alpha\_init
  - teb::Config::Optim::Solver::NonlinearProgram::LineSearch, 174
- applyIncrement
  - teb::BaseSolver, 70
  - teb::BaseSolverLeastSquares, 79
  - teb::BaseSolverNonlinearProgramDense, 90
  - teb::SolverLevenbergMarquardtEigenDense, 205
  - teb::SolverLevenbergMarquardtEigenSparse, 217
  - teb::SolverNloptPackage, 229
  - teb::SolverSQPDense, 241
- applyOptVec
  - teb::BaseSolver, 70
  - teb::BaseSolverLeastSquares, 79
  - teb::BaseSolverNonlinearProgramDense, 90
  - teb::SolverLevenbergMarquardtEigenDense, 205
  - teb::SolverLevenbergMarquardtEigenSparse, 218
  - teb::SolverNloptPackage, 230
  - teb::SolverSQPDense, 242
- BLOCK\_BFGS
  - teb, 43
- BOUND\_TYPE
  - teb, 42
- BOUND\_VARS
  - teb, 42
- backupJacobian
  - teb::BaseEdge, 52
  - teb::BaseEdge< D, FuncType >, 61
  - teb::BoundConstraint, 104
  - teb::EdgeMinimizeTime, 129
  - teb::EdgeQuadraticForm, 138
  - teb::EdgeSystemDynamics, 148
  - teb::EdgeType, 156
- BackupStackType
  - teb, 41
  - teb::ControlVertex, 116
  - teb::StateVertex, 258
- backupVertices
  - teb::BaseSolver, 70
  - teb::BaseSolverLeastSquares, 79
  - teb::BaseSolverNonlinearProgramDense, 91
  - teb::SolverLevenbergMarquardtEigenDense, 205
  - teb::SolverLevenbergMarquardtEigenSparse, 218
  - teb::SolverNloptPackage, 230
  - teb::SolverSQPDense, 242
- base\_controller.h, 323
- base\_edge.h, 323
- base\_edge.hpp, 324
- base\_edge\_dynamics.h, 324
- base\_solver.h, 324
- base\_solver\_least\_squares.cpp, 325
- base\_solver\_least\_squares.h, 325
- base\_solver\_nonlinear\_program\_dense.cpp, 325
- base\_solver\_nonlinear\_program\_dense.h, 325
- BaseEdge
  - teb::BaseEdge, 52

- teb::BaseEdge< D, FuncType >, 61
- BaseSolver
  - teb::BaseSolver, 69
- BaseSolverLeastSquares
  - teb::BaseSolverLeastSquares, 78
- BaseSolverNonlinearProgramDense
  - teb::BaseSolverNonlinearProgramDense, 90
- beta
  - teb::Config::Optim::Solver::NonlinearProgram::LineSearch, 174
- bfgs\_damped\_mode
  - teb::Config::Optim::Solver::NonlinearProgram::Hessian, 162
- bound\_constraints.h, 326
- BoundConstraint
  - teb::BoundConstraint, 104
- BoundType
  - teb::BoundConstraint, 111
- buildEqualityConstraintJacobian
  - teb::BaseSolverNonlinearProgramDense, 91
  - teb::SolverSQPDense, 242
- buildEqualityConstraintValueVector
  - teb::BaseSolverNonlinearProgramDense, 91
  - teb::SolverSQPDense, 242
- buildInequalityConstraintJacobian
  - teb::BaseSolverNonlinearProgramDense, 91
  - teb::SolverSQPDense, 242
- buildInequalityConstraintValueVector
  - teb::BaseSolverNonlinearProgramDense, 91
  - teb::SolverSQPDense, 243
- buildJacobian
  - teb::SolverLevenbergMarquardtEigenDense, 206
  - teb::SolverLevenbergMarquardtEigenSparse, 218
- buildObjectiveGradient
  - teb::BaseSolverNonlinearProgramDense, 91
  - teb::SolverSQPDense, 243
- buildObjectiveValue
  - teb::BaseSolverNonlinearProgramDense, 92
  - teb::SolverSQPDense, 243
- buildOptimizationGraph
  - teb::TebController, 280
- buildValueVector
  - teb::BaseSolverLeastSquares, 80
  - teb::SolverLevenbergMarquardtEigenDense, 206
  - teb::SolverLevenbergMarquardtEigenSparse, 219
- CENTRAL
  - teb, 42
- calculateLagrangianGradient
  - teb::BaseSolverNonlinearProgramDense, 92
  - teb::SolverSQPDense, 243
- calculateLagrangianHessian
  - teb::BaseSolverNonlinearProgramDense, 92
  - teb::SolverSQPDense, 243
- calculateLagrangianHessianFullBFGS
  - teb::BaseSolverNonlinearProgramDense, 92
  - teb::SolverSQPDense, 244
- calculateLagrangianHessianNumerically
  - teb::BaseSolverNonlinearProgramDense, 92
- teb::SolverSQPDense, 244
- calculateMerit
  - teb::SolverSQPDense, 244
- calculateMeritDerivative
  - teb::SolverSQPDense, 244
- calculateVertexDimensions
  - teb::BaseEdge, 52
  - teb::BaseEdge< D, FuncType >, 62
  - teb::BoundConstraint, 104
  - teb::EdgeMinimizeTime, 129
  - teb::EdgeQuadraticForm, 138
  - teb::EdgeSystemDynamics, 148
- cfg
  - teb::BaseSolver, 75
  - teb::BaseSolverLeastSquares, 86
  - teb::BaseSolverNonlinearProgramDense, 100
  - teb::SolverLevenbergMarquardtEigenDense, 213
  - teb::SolverLevenbergMarquardtEigenSparse, 226
  - teb::SolverNloptPackage, 236
  - teb::SolverSQPDense, 255
  - teb::TebController, 299
- checkConvergence
  - teb::SolverSQPDense, 245
- clearActiveVertices
  - teb::HyperGraph, 169
- clearEdges
  - teb::HyperGraph, 169
- clearGraph
  - teb::HyperGraph, 169
- clearWindow
  - teb::TebPlotter, 302
- closeWindow
  - teb::TebPlotter, 302
- common\_teb\_edges.h, 326
- completePdfExport
  - teb::TebPlotter, 302
- computeHessian
  - teb::BaseEdge, 53
  - teb::BaseEdge< D, FuncType >, 62
  - teb::BoundConstraint, 104
  - teb::EdgeMinimizeTime, 129
  - teb::EdgeQuadraticForm, 138
  - teb::EdgeSystemDynamics, 148
  - teb::EdgeType, 156
- computeJacobian
  - teb::BaseEdge, 53
  - teb::BaseEdge< D, FuncType >, 62
  - teb::BoundConstraint, 105
  - teb::EdgeMinimizeTime, 129
  - teb::EdgeQuadraticForm, 138
  - teb::EdgeSystemDynamics, 148
  - teb::EdgeType, 156
- computeValues
  - teb::BaseEdge, 54
  - teb::BaseEdge< D, FuncType >, 63
  - teb::BoundConstraint, 105
  - teb::EdgeMinimizeTime, 130
  - teb::EdgeQuadraticForm, 139



- teb::EdgeSystemDynamics, 149
- teb::EdgeType, 156
- config.h, 327
- conservativeResize
  - teb::SimResults, 189
- constraintsEq
  - teb::SolverNloptPackage, 230
- constraintsInEq
  - teb::SolverNloptPackage, 230
- ControlSequence
  - teb::TebController, 276
- controlSequence
  - teb::TebController, 281
- ControlVector
  - teb::ControlVertex, 116
  - teb::ExplicitEuler, 160
  - teb::NumericalIntegrator, 180
  - teb::RungeKutta5thOrder, 184
  - teb::RungeKuttaClassic, 186
  - teb::Simulator, 192
  - teb::SystemDynamics, 267
  - teb::TebController, 276
- ControlVertex
  - teb::ControlVertex, 117
- Controller, 33
  - EdgeControlBounds, 34
  - EdgePositiveTime, 35
  - EdgeStateBounds, 35
- controls
  - teb::ControlVertex, 117
  - teb::SimResults::TimeSeries, 316
- countJacobianColNNZ
  - teb::SolverLevenbergMarquardtEigenSparse, 219
- customOptimizationGraph
  - teb::TebController, 281
- customOptimizationGraphHotStart
  - teb::TebController, 281
- diff\_method
  - teb::Config::Teb, 270
- DimControls
  - teb::ControlVertex, 124
  - teb::SystemDynamics, 269
- DimStates
  - teb::StateVertex, 265
  - teb::SystemDynamics, 269
- Dimension
  - teb::BaseEdge, 57
  - teb::BaseEdge< D, FuncType >, 67
  - teb::BoundConstraint, 111
  - teb::ControlVertex, 124
  - teb::EdgeMinimizeTime, 133
  - teb::EdgeQuadraticForm, 143
  - teb::EdgeSystemDynamics, 153
  - teb::StateVertex, 265
  - teb::TimeDiff, 315
- dimension
  - teb::BaseEdge, 54
  - teb::BaseEdge< D, FuncType >, 63
- teb::BoundConstraint, 105
- teb::ControlVertex, 117
- teb::EdgeMinimizeTime, 130
- teb::EdgeQuadraticForm, 139
- teb::EdgeSystemDynamics, 149
- teb::EdgeType, 156
- teb::StateVertex, 259
- teb::TimeDiff, 311
- teb::VertexType, 318
- dimensionFree
  - teb::ControlVertex, 118
  - teb::StateVertex, 259
  - teb::TimeDiff, 311
  - teb::VertexType, 318
- discardBackupVertices
  - teb::BaseSolver, 70
  - teb::BaseSolverLeastSquares, 80
  - teb::BaseSolverNonlinearProgramDense, 92
  - teb::SolverLevenbergMarquardtEigenDense, 207
  - teb::SolverLevenbergMarquardtEigenSparse, 219
  - teb::SolverNloptPackage, 230
  - teb::SolverSQPDense, 245
- discardJacobianBackup
  - teb::BaseEdge, 54
  - teb::BaseEdge< D, FuncType >, 63
  - teb::BoundConstraint, 105
  - teb::EdgeMinimizeTime, 130
  - teb::EdgeQuadraticForm, 139
  - teb::EdgeSystemDynamics, 149
  - teb::EdgeType, 156
- discardTop
  - teb::ControlVertex, 118
  - teb::StateVertex, 259
  - teb::TimeDiff, 311
  - teb::VertexType, 318
- download.md, 328
- dt
  - teb::SimResults::TimeSeries, 316
  - teb::TebController, 282
  - teb::TimeDiff, 311
- dt\_hyst
  - teb::Config::Teb, 270
- dt\_min
  - teb::Config::Teb, 270
- dt\_ref
  - teb::Config::Teb, 270
- EPS
  - teb::TebPlotter, 302
- EXPLICIT\_EULER
  - teb, 43
- EdgeContainer
  - teb::BaseEdge, 51
  - teb::BaseEdge< D, FuncType >, 61
  - teb::BaseSolver, 69
  - teb::BaseSolverLeastSquares, 78
  - teb::BaseSolverNonlinearProgramDense, 90
  - teb::BoundConstraint, 103
  - teb::EdgeMinimizeTime, 128

- teb::EdgeQuadraticForm, [137](#)
- teb::EdgeSystemDynamics, [147](#)
- teb::EdgeType, [155](#)
- teb::HyperGraph, [167](#)
- teb::SolverLevenbergMarquardtEigenDense, [204](#)
- teb::SolverLevenbergMarquardtEigenSparse, [216](#)
- teb::SolverNloptPackage, [229](#)
- teb::SolverSQPDense, [240](#)
- teb::TebController, [276](#)
- EdgeControlBounds
  - Controller, [34](#)
- EdgeMinimizeTime
  - teb::EdgeMinimizeTime, [129](#)
- EdgePositiveTime
  - Controller, [35](#)
- EdgeQuadraticForm
  - teb::EdgeQuadraticForm, [137](#)
- EdgeStateBounds
  - Controller, [35](#)
- EdgeSystemDynamics
  - teb::EdgeSystemDynamics, [147](#)
  - teb::SystemDynamics, [269](#)
- EdgeType
  - teb::EdgeType, [155](#)
- EigenScalar
  - teb, [41](#)
- EigenScalarD
  - teb, [41](#)
- equalities
  - teb::HyperGraph, [169](#)
- equalityConstraintFunction
  - teb::SolverNloptPackage, [230](#)
- ExplicitEuler
  - teb::ExplicitEuler, [160](#)
- FORWARD
  - teb, [42](#)
- FULL\_BFGS
  - teb, [43](#)
- FULL\_BFGS\_WITH\_STRUCTURE\_FILTER
  - teb, [43](#)
- FUNCT\_TYPE
  - teb, [42](#)
- FileFormat
  - teb::TebPlotter, [301](#)
- FiniteDifferences
  - teb, [42](#)
- finiteElemCentralDiff
  - teb::SystemDynamics, [267](#)
- finiteElemFwdEuler
  - teb::SystemDynamics, [268](#)
- firstControl
  - teb::BaseController, [46](#)
  - teb::TebController, [282](#)
- firstControlRef
  - teb::TebController, [282](#)
- firstControlSaturated
  - teb::TebController, [282](#)
- firstState
  - teb::BaseController, [46](#)
  - teb::TebController, [283](#)
- firstStateRef
  - teb::TebController, [283](#)
- fixed\_ctrls
  - teb::ControlVertex, [118](#)
- fixed\_states
  - teb::StateVertex, [259](#)
- FixedCtrls
  - teb::ControlVertex, [116](#)
- FixedStates
  - teb::StateVertex, [258](#)
  - teb::TebController, [276](#)
- force\_rebuild\_optim\_graph
  - teb::Config::Optim, [181](#)
- functType
  - teb::BaseEdge, [54](#)
  - teb::BaseEdge< D, FuncType >, [63](#)
  - teb::BoundConstraint, [105](#)
  - teb::EdgeMinimizeTime, [130](#)
  - teb::EdgeQuadraticForm, [139](#)
  - teb::EdgeSystemDynamics, [149](#)
  - teb::EdgeType, [156](#)
- getAbsoluteTimeVec
  - teb::BaseController, [46](#)
  - teb::TebController, [283](#)
- getActiveVertices
  - teb::TebController, [283](#)
- getBoundConstrDimFromStorage
  - teb::SolverNloptPackage, [231](#)
- getBounds
  - teb::BoundConstraint, [105](#)
- getChi2
  - teb::BaseSolverLeastSquares, [80](#)
  - teb::SolverLevenbergMarquardtEigenDense, [207](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [220](#)
- getCustomData
  - teb::BaseEdge, [54](#)
  - teb::BaseEdge< D, FuncType >, [63](#)
  - teb::BoundConstraint, [106](#)
  - teb::EdgeMinimizeTime, [130](#)
  - teb::EdgeQuadraticForm, [139](#)
  - teb::EdgeSystemDynamics, [149](#)
  - teb::EdgeType, [157](#)
- getData
  - teb::ControlVertex, [118](#)
  - teb::StateVertex, [260](#)
  - teb::TimeDiff, [311](#)
  - teb::VertexType, [319](#)
- getDataFree
  - teb::ControlVertex, [118](#)
  - teb::StateVertex, [260](#)
  - teb::TimeDiff, [311](#)
  - teb::VertexType, [319](#)
- getDimEqualities
  - teb::BaseSolver, [70](#)
  - teb::BaseSolverLeastSquares, [81](#)
  - teb::BaseSolverNonlinearProgramDense, [93](#)

- teb::SolverLevenbergMarquardtEigenDense, 207
- teb::SolverLevenbergMarquardtEigenSparse, 220
- teb::SolverNloptPackage, 231
- teb::SolverSQPDense, 245
- getDimInequalities
  - teb::BaseSolver, 71
  - teb::BaseSolverLeastSquares, 81
  - teb::BaseSolverNonlinearProgramDense, 93
  - teb::SolverLevenbergMarquardtEigenDense, 207
  - teb::SolverLevenbergMarquardtEigenSparse, 220
  - teb::SolverNloptPackage, 231
  - teb::SolverSQPDense, 245
- getDimObjectives
  - teb::BaseSolver, 71
  - teb::BaseSolverLeastSquares, 81
  - teb::BaseSolverNonlinearProgramDense, 93
  - teb::SolverLevenbergMarquardtEigenDense, 208
  - teb::SolverLevenbergMarquardtEigenSparse, 220
  - teb::SolverNloptPackage, 231
  - teb::SolverSQPDense, 245
- getDt
  - teb::BaseController, 46
  - teb::TebController, 284
- getEqConstrDimFromStorage
  - teb::SolverNloptPackage, 231
- getInEqConstrDimFromStorage
  - teb::SolverNloptPackage, 231
- getJacobianBackup
  - teb::BaseEdge, 54
  - teb::BaseEdge< D, FuncType >, 63
  - teb::BoundConstraint, 106
  - teb::EdgeMinimizeTime, 130
  - teb::EdgeQuadraticForm, 139
  - teb::EdgeSystemDynamics, 149
  - teb::EdgeType, 157
- getM
  - teb::TebController, 284
- getN
  - teb::BaseController, 47
  - teb::TebController, 284
- getOptVecCopy
  - teb::BaseSolver, 71
  - teb::BaseSolverLeastSquares, 81
  - teb::BaseSolverNonlinearProgramDense, 93
  - teb::SolverLevenbergMarquardtEigenDense, 208
  - teb::SolverLevenbergMarquardtEigenSparse, 220
  - teb::SolverNloptPackage, 231
  - teb::SolverSQPDense, 245
- getOptVecDimFromStorage
  - teb::SolverNloptPackage, 232
- getOptVecDimension
  - teb::BaseSolver, 71
  - teb::BaseSolverLeastSquares, 81
  - teb::BaseSolverNonlinearProgramDense, 93
  - teb::SolverLevenbergMarquardtEigenDense, 208
  - teb::SolverLevenbergMarquardtEigenSparse, 221
  - teb::SolverNloptPackage, 232
  - teb::SolverSQPDense, 246
- getOptVecIdx
  - teb::ControlVertex, 118
  - teb::StateVertex, 260
  - teb::TimeDiff, 312
  - teb::VertexType, 319
- getSampleOptVecIdxVMat
  - teb::TebController, 284
- getStateCtrlInfoMat
  - teb::BaseController, 47
  - teb::TebController, 285
- getTEBFixedMap
  - teb::TebController, 285
- getValueDimension
  - teb::BaseSolverLeastSquares, 82
  - teb::SolverLevenbergMarquardtEigenDense, 208
  - teb::SolverLevenbergMarquardtEigenSparse, 221
- getVertex
  - teb::BaseEdge, 54, 55
  - teb::BaseEdge< D, FuncType >, 64
  - teb::BoundConstraint, 106
  - teb::EdgeMinimizeTime, 130
  - teb::EdgeQuadraticForm, 140
  - teb::EdgeSystemDynamics, 150
  - teb::EdgeType, 157
- getWorkspace
  - teb::HessianWorkspace, 164
  - teb::JacobianWorkspace, 173
- goal\_dist\_force\_reinit
  - teb::Config::Teb, 270
- graph
  - teb::TebController, 285
- graph.h, 328
- groups.dox, 329
- hessian
  - teb::Config::Optim::Solver::NonlinearProgram, 178
- hessian\_init
  - teb::Config::Optim::Solver::NonlinearProgram::-Hessian, 162
- hessian\_init\_identity\_scale
  - teb::Config::Optim::Solver::NonlinearProgram::-Hessian, 162
- hessian\_method
  - teb::Config::Optim::Solver::NonlinearProgram::-Hessian, 162
- HessianInit
  - teb, 42
- HessianMethod
  - teb, 42
- HessianWorkspace
  - teb::HessianWorkspace, 164
- hessians
  - teb::BaseEdge, 55
  - teb::BaseEdge< D, FuncType >, 64
  - teb::BoundConstraint, 107
  - teb::EdgeMinimizeTime, 131
  - teb::EdgeQuadraticForm, 140
  - teb::EdgeSystemDynamics, 150
  - teb::EdgeType, 157

- IDENTITY
  - teb, [42](#)
- INF
  - typedefs.h, [339](#)
- INPUT\_STREAM
  - typedefs.h, [339](#)
- index
  - teb::CustomBoundData, [125](#)
- inequalities
  - teb::HyperGraph, [170](#)
- inequalityConstraintFunction
  - teb::SolverNloptPackage, [232](#)
- initHessianBFGS
  - teb::BaseSolverNonlinearProgramDense, [94](#)
  - teb::SolverSQPDense, [246](#)
- initOptimization
  - teb::TebController, [286](#)
- initSolverWorkspace
  - teb::BaseSolverNonlinearProgramDense, [94](#)
  - teb::SolverSQPDense, [246](#)
- initTrajectory
  - teb::TebController, [286](#)
- initWorkspaces
  - teb::BaseSolver, [71](#)
  - teb::BaseSolverLeastSquares, [82](#)
  - teb::BaseSolverNonlinearProgramDense, [94](#)
  - teb::SolverLevenbergMarquardtEigenDense, [209](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [221](#)
  - teb::SolverNloptPackage, [232](#)
  - teb::SolverSQPDense, [247](#)
- initializeLagrangeMultiplier
  - teb::BaseSolverNonlinearProgramDense, [94](#)
  - teb::SolverSQPDense, [246](#)
- insertStateControlInputPair
  - teb::TebController, [286](#)
- install.md, [329](#)
- IntegratorPtr
  - teb::Simulator, [192](#)
- integrate
  - teb::ExplicitEuler, [160](#)
  - teb::NumericalIntegrator, [180](#)
  - teb::RungeKutta5thOrder, [184](#)
  - teb::RungeKuttaClassic, [187](#)
- integrators.h, [329](#)
- isBoundConstraint
  - teb::BaseEdge, [55](#)
  - teb::BaseEdge< D, FuncType >, [64](#)
  - teb::BoundConstraint, [107](#)
  - teb::EdgeMinimizeTime, [131](#)
  - teb::EdgeQuadraticForm, [140](#)
  - teb::EdgeSystemDynamics, [150](#)
  - teb::EdgeType, [157](#)
- isExportToFileEnabled
  - teb::TebPlotter, [302](#)
- isFixedAll
  - teb::ControlVertex, [119](#)
  - teb::StateVertex, [260](#)
  - teb::TimeDiff, [312](#)
- teb::VertexType, [319](#)
- isFixedAny
  - teb::ControlVertex, [119](#)
  - teb::StateVertex, [260](#)
  - teb::TimeDiff, [312](#)
  - teb::VertexType, [319](#)
- isFixedComp
  - teb::ControlVertex, [119](#)
  - teb::StateVertex, [260](#)
  - teb::TimeDiff, [312](#)
  - teb::VertexType, [319](#)
- isGraphModified
  - teb::HyperGraph, [170](#)
- JPEG
  - teb::TebPlotter, [301](#)
- JacobianWorkspace
  - teb::JacobianWorkspace, [173](#)
- jacobians
  - teb::BaseEdge, [55](#)
  - teb::BaseEdge< D, FuncType >, [64](#)
  - teb::BoundConstraint, [107](#)
  - teb::EdgeMinimizeTime, [131](#)
  - teb::EdgeQuadraticForm, [140](#)
  - teb::EdgeSystemDynamics, [150](#)
  - teb::EdgeType, [157](#), [158](#)
- LINEAR
  - teb, [42](#)
- LINEAR\_SQUARED
  - teb, [42](#)
- LOWER
  - teb, [42](#)
- LOWERUPPER
  - teb, [42](#)
- lastState
  - teb::BaseController, [47](#)
  - teb::TebController, [287](#)
- lastStateRef
  - teb::TebController, [287](#)
- legend
  - teb::PlotOptions, [182](#)
- legend\_entries
  - teb::PlotOptions, [182](#)
- linsearch
  - teb::Config::Optim::Solver::NonlinearProgram, [178](#)
- lower
  - teb::CustomBoundData, [125](#)
- lsq
  - teb::Config::Optim::Solver, [201](#)
- mainpage.dox, [329](#)
- MatMapRowMajor
  - teb::BaseSolverNonlinearProgramDense, [90](#)
  - teb::SolverSQPDense, [240](#)
- matlab\_class\_handle.hpp, [329](#)
- matlab\_interface.md, [329](#)
- max\_optimization\_time
  - teb::Config::Optim::Solver::Nlopt, [177](#)

measure\_cpu\_time.h, [330](#)  
     START\_TIMER, [330](#)  
     STOP\_TIMER, [330](#)  
 misc.h, [330](#)  
     norm\_angle, [331](#)  
     norm\_angle\_vec, [332](#)  
  
 NONLINEAR  
     teb, [42](#)  
 NONLINEAR\_ONCE\_DIFF  
     teb, [42](#)  
 NONLINEAR\_SQUARED  
     teb, [42](#)  
 NUMERIC  
     teb, [42](#), [43](#)  
 n\_max  
     teb::Config::Teb, [270](#)  
 n\_min  
     teb::Config::Teb, [270](#)  
 n\_pre  
     teb::Config::Teb, [270](#)  
 nlopt  
     teb::Config::Optim::Solver, [201](#)  
 NloptAlgorithms  
     teb, [43](#)  
 no\_lower  
     teb::CustomBoundData, [125](#)  
 no\_upper  
     teb::CustomBoundData, [125](#)  
 NoBounds  
     teb::BoundConstraint, [112](#)  
 NoControls  
     teb::TebController, [299](#)  
 NoStates  
     teb::TebController, [299](#)  
 NoVertices  
     teb::BaseEdge, [57](#)  
     teb::BoundConstraint, [112](#)  
     teb::EdgeMinimizeTime, [133](#)  
     teb::EdgeQuadraticForm, [143](#)  
     teb::EdgeSystemDynamics, [153](#)  
 noVertices  
     teb::BaseEdge, [55](#)  
     teb::BaseEdge< D, FuncType >, [64](#)  
     teb::BoundConstraint, [107](#)  
     teb::EdgeMinimizeTime, [131](#)  
     teb::EdgeQuadraticForm, [140](#)  
     teb::EdgeSystemDynamics, [150](#)  
     teb::EdgeType, [158](#)  
 nonlin\_prog  
     teb::Config::Optim::Solver, [201](#)  
 norm\_angle  
     misc.h, [331](#)  
 norm\_angle\_vec  
     misc.h, [332](#)  
 notifyGraphModified  
     teb::HyperGraph, [170](#)  
 NumericalIntegrator  
     teb::NumericalIntegrator, [180](#)  
  
 NumericalIntegrators  
     teb, [43](#)  
  
 objectiveFunction  
     teb::SolverNloptPackage, [232](#)  
 objectives  
     teb::HyperGraph, [170](#), [171](#)  
     teb::SolverNloptPackage, [232](#)  
 operator<<  
     teb::ControlVertex, [123](#)  
     teb::StateVertex, [264](#)  
     teb::TimeDiff, [314](#)  
 operator()  
     teb::TimeDiff, [312](#)  
 operator=  
     teb::ControlVertex, [119](#)  
     teb::StateVertex, [261](#)  
     teb::TimeDiff, [312](#)  
 operator==  
     teb::ControlVertex, [123](#)  
     teb::StateVertex, [264](#)  
 optim  
     teb::Config, [113](#)  
 optimizeTEB  
     teb::TebController, [287](#)  
  
 PDF  
     teb::TebPlotter, [302](#)  
 PNG  
     teb::TebPlotter, [301](#)  
 PI  
     typedefs.h, [339](#)  
 PRINT\_DEBUG  
     typedefs.h, [339](#)  
 PRINT\_DEBUG\_COND  
     typedefs.h, [339](#)  
 PRINT\_DEBUG\_ONCE  
     typedefs.h, [339](#)  
 PRINT\_ERROR  
     typedefs.h, [340](#)  
 PRINT\_INFO  
     typedefs.h, [340](#)  
 PRINT\_INFO\_COND  
     typedefs.h, [340](#)  
 PRINT\_INFO\_ONCE  
     typedefs.h, [340](#)  
 pipe  
     teb::TebPlotter, [307](#)  
 plot  
     teb::TebPlotter, [303](#)  
 plot1DVector  
     teb::TebPlotter, [303](#)  
 plotCustomKey  
     teb::TebPlotter, [303](#)  
 plotMulti  
     teb::TebPlotter, [304](#)  
 plotResults  
     teb::Simulator, [193](#)  
 plotTEB

- teb::TebPlotter, [304](#), [305](#)
- plotTwoCol
  - teb::TebPlotter, [305](#)
- plus
  - teb::ControlVertex, [119](#)
  - teb::StateVertex, [261](#)
  - teb::TimeDiff, [313](#)
  - teb::VertexType, [320](#)
- plusFree
  - teb::ControlVertex, [120](#)
  - teb::StateVertex, [261](#)
  - teb::TimeDiff, [313](#)
  - teb::VertexType, [320](#)
- pop
  - teb::ControlVertex, [120](#)
  - teb::StateVertex, [261](#)
  - teb::TimeDiff, [313](#)
  - teb::VertexType, [320](#)
- predictControl
  - teb::TebController, [287](#)
- push
  - teb::ControlVertex, [120](#)
  - teb::StateVertex, [261](#)
  - teb::TimeDiff, [313](#)
  - teb::VertexType, [320](#)
- pushBackStateControlInputPair
  - teb::TebController, [288](#)
- pushFrontStateControlInputPair
  - teb::TebController, [288](#)
- QUADRATIC
  - teb, [42](#)
- RUNGE\_KUTTA\_5TH
  - teb, [43](#)
- RUNGE\_KUTTA\_CLASSIC
  - teb, [43](#)
- removeStateControlInputPair
  - teb::TebController, [288](#)
- resampleTrajectory
  - teb::TebController, [289](#)
- resetController
  - teb::BaseController, [47](#)
  - teb::TebController, [289](#)
- resizeTrajectory
  - teb::TebController, [289](#)
- resizeVertexContainer
  - teb::BaseEdge< D, FuncType >, [65](#)
- restoreJacobian
  - teb::BaseEdge, [55](#)
  - teb::BaseEdge< D, FuncType >, [65](#)
  - teb::BoundConstraint, [107](#)
  - teb::EdgeMinimizeTime, [131](#)
  - teb::EdgeQuadraticForm, [141](#)
  - teb::EdgeSystemDynamics, [151](#)
  - teb::EdgeType, [158](#)
- restoreVertices
  - teb::BaseSolver, [72](#)
  - teb::BaseSolverLeastSquares, [82](#)
  - teb::BaseSolverNonlinearProgramDense, [94](#)
  - teb::SolverLevenbergMarquardtEigenDense, [209](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [221](#)
  - teb::SolverNloptPackage, [232](#)
  - teb::SolverSQPDense, [247](#)
- restoreVerticesButKeepBackup
  - teb::BaseSolver, [72](#)
  - teb::BaseSolverLeastSquares, [82](#)
  - teb::BaseSolverNonlinearProgramDense, [95](#)
  - teb::SolverLevenbergMarquardtEigenDense, [209](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [222](#)
  - teb::SolverNloptPackage, [233](#)
  - teb::SolverSQPDense, [247](#)
- returnControlInputSequence
  - teb::BaseController, [47](#)
  - teb::TebController, [290](#)
- RungeKutta5thOrder
  - teb::RungeKutta5thOrder, [184](#)
- RungeKuttaClassic
  - teb::RungeKuttaClassic, [186](#)
- SINGLE
  - teb, [42](#)
- SLSQP
  - teb, [43](#)
- SVG
  - teb::TebPlotter, [301](#)
- START\_TIMER
  - measure\_cpu\_time.h, [330](#)
- STOP\_TIMER
  - measure\_cpu\_time.h, [330](#)
- sampleTime
  - teb::Simulator, [193](#)
- saturateControl
  - teb::Simulator, [194](#)
- ScalarType
  - teb::ControlVertex, [116](#)
  - teb::StateVertex, [258](#)
- seperateInequalitiesAndBounds
  - teb::SolverNloptPackage, [233](#)
- series
  - teb::SimResults, [189](#)
- setBounds
  - teb::BoundConstraint, [107–109](#)
- setConfig
  - teb::BaseSolver, [72](#)
  - teb::BaseSolverLeastSquares, [83](#)
  - teb::BaseSolverNonlinearProgramDense, [95](#)
  - teb::SolverLevenbergMarquardtEigenDense, [209](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [222](#)
  - teb::SolverNloptPackage, [233](#)
  - teb::SolverSQPDense, [247](#)
- setControlInputSaturation
  - teb::Simulator, [194](#)
- setControls
  - teb::ControlVertex, [120](#)
- setData
  - teb::EdgeMinimizeTime, [131](#)
- setFixedAll



- teb::ControlVertex, [120](#)
- teb::StateVertex, [262](#)
- teb::TimeDiff, [313](#)
- setFixedCtrl
  - teb::ControlVertex, [121](#)
- setFixedCtrls
  - teb::ControlVertex, [121](#)
- setFixedDt
  - teb::TebController, [290](#)
- setFixedGoal
  - teb::TebController, [290](#), [291](#)
- setFixedStart
  - teb::TebController, [291](#)
- setFixedState
  - teb::StateVertex, [262](#)
- setFixedStates
  - teb::StateVertex, [262](#)
- setFree
  - teb::ControlVertex, [121](#)
  - teb::StateVertex, [263](#)
  - teb::TimeDiff, [313](#)
  - teb::VertexType, [320](#)
- setGoalStatesFixedOrUnfixed
  - teb::TebController, [291](#), [293](#)
- setIntegrator
  - teb::Simulator, [194](#)
- setOptVecIdx
  - teb::ControlVertex, [121](#)
  - teb::StateVertex, [263](#)
  - teb::TimeDiff, [314](#)
  - teb::VertexType, [320](#)
- setOutputToFile
  - teb::TebPlotter, [306](#)
- setOutputToWindow
  - teb::TebPlotter, [306](#)
- setPlotter
  - teb::Simulator, [194](#)
- setPostStepCallback
  - teb::Simulator, [195](#)
- setPreSimCallback
  - teb::Simulator, [195](#)
- setPreStepCallback
  - teb::Simulator, [195](#)
- setReference
  - teb::EdgeQuadraticForm, [141](#)
- setSampleTime
  - teb::Simulator, [196](#)
- setSolver
  - teb::TebController, [293](#)
- setStates
  - teb::StateVertex, [263](#)
- setSystemDynamics
  - teb::EdgeSystemDynamics, [151](#)
  - teb::TebController, [294](#)
- setVertex
  - teb::BaseEdge< D, FuncType >, [65](#)
- setWeights
  - teb::EdgeQuadraticForm, [141](#)
- setupHorizon
  - teb::TebController, [294](#)
- sigma
  - teb::Config::Optim::Solver::NonlinearProgram::-LineSearch, [175](#)
- simClosedLoop
  - teb::Simulator, [196](#), [197](#)
- simOpenAndClosedLoop
  - teb::Simulator, [197](#)
- simOpenLoop
  - teb::Simulator, [198](#)
- SimResults
  - teb::Simulator, [199](#)
- Simulation, [37](#)
- Simulator
  - teb::SimResults, [189](#)
  - teb::Simulator, [193](#)
  - teb::SystemDynamics, [269](#)
- simulator.h, [332](#)
- simulator.hpp, [333](#)
- skip\_last\_value\_right\_column
  - teb::PlotOptions, [182](#)
- soft\_constr\_epsilon
  - teb::Config::Optim::Solver::Lsq, [175](#)
- solve
  - teb::BaseSolver, [72](#)
  - teb::BaseSolverLeastSquares, [83](#)
  - teb::BaseSolverNonlinearProgramDense, [95](#)
  - teb::SolverLevenbergMarquardtEigenDense, [209](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [222](#)
  - teb::SolverNloptPackage, [233](#)
  - teb::SolverSQPDense, [248](#)
- solveImpl
  - teb::BaseSolver, [73](#)
  - teb::BaseSolverLeastSquares, [83](#)
  - teb::BaseSolverNonlinearProgramDense, [95](#)
  - teb::SolverLevenbergMarquardtEigenDense, [210](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [222](#)
  - teb::SolverNloptPackage, [233](#)
  - teb::SolverSQPDense, [248](#)
- Solver, [36](#)
- solver
  - teb::Config::Optim, [181](#)
- solver\_iter
  - teb::Config::Optim::Solver, [201](#)
- solver\_levenbergmarquardt\_eigen\_dense.cpp, [333](#)
- solver\_levenbergmarquardt\_eigen\_dense.h, [333](#)
- solver\_levenbergmarquardt\_eigen\_sparse.cpp, [333](#)
- solver\_levenbergmarquardt\_eigen\_sparse.h, [334](#)
- solver\_nlopt\_package.cpp, [334](#)
- solver\_nlopt\_package.h, [334](#)
- solver\_overview.md, [334](#)
- solver\_sqp\_dense.h, [334](#)
- solver\_sqp\_dense\_backup.h, [335](#)
- SolverLevenbergMarquardtEigenDense
  - teb::SolverLevenbergMarquardtEigenDense, [204](#)
- SolverLevenbergMarquardtEigenSparse
  - teb::SolverLevenbergMarquardtEigenSparse, [217](#)

- SolverNloptPackage
  - teb::SolverNloptPackage, 229
- SolverSQPDense
  - teb::SolverSQPDense, 240
- spyMatrix
  - teb::TebPlotter, 306
- stackSize
  - teb::ControlVertex, 123
  - teb::StateVertex, 263
  - teb::TimeDiff, 314
  - teb::VertexType, 320
- StateSequence
  - teb::TebController, 276
- stateSequence
  - teb::TebController, 295
- stateSpaceModel
  - teb::SystemDynamics, 268
- StateVector
  - teb::EdgeQuadraticForm, 137
  - teb::ExplicitEuler, 160
  - teb::NumericalIntegrator, 180
  - teb::RungeKutta5thOrder, 184
  - teb::RungeKuttaClassic, 186
  - teb::Simulator, 192
  - teb::StateVertex, 258
  - teb::SystemDynamics, 267
  - teb::TebController, 276
- StateVertex
  - teb::StateVertex, 259
- states
  - teb::SimResults::TimeSeries, 316
  - teb::StateVertex, 263, 264
- step
  - teb::BaseController, 47
  - teb::TebController, 295
- stopping\_criteria\_ftol\_abs
  - teb::Config::Optim::Solver::Nlopt, 177
- stopping\_criteria\_ftol\_rel
  - teb::Config::Optim::Solver::Nlopt, 177
- stopping\_criteria\_xtol\_abs
  - teb::Config::Optim::Solver::Nlopt, 177
- stopping\_criteria\_xtol\_rel
  - teb::Config::Optim::Solver::Nlopt, 177
- switchWindow
  - teb::TebPlotter, 306
- system\_dynamics.h, 335
- SystemDynamics
  - teb::EdgeSystemDynamics, 152
  - teb::Simulator, 199
  - teb::SystemDynamics, 267
- systemStep
  - teb::Simulator, 198
- TIKZ
  - teb::TebPlotter, 302
- teb, 39
  - ALL, 42
  - BLOCK\_BFGS, 43
  - BOUND\_TYPE, 42
  - BOUND\_VARS, 42
  - BackupStackType, 41
  - CENTRAL, 42
  - EXPLICIT\_EULER, 43
  - EigenScalar, 41
  - EigenScalarD, 41
  - FORWARD, 42
  - FULL\_BFGS, 43
  - FULL\_BFGS\_WITH\_STRUCTURE\_FILTER, 43
  - FUNCT\_TYPE, 42
  - FiniteDifferences, 42
  - HessianInit, 42
  - HessianMethod, 42
  - IDENTITY, 42
  - LINEAR, 42
  - LINEAR\_SQUARED, 42
  - LOWER, 42
  - LOWERUPPER, 42
  - NONLINEAR, 42
  - NONLINEAR\_ONCE\_DIFF, 42
  - NONLINEAR\_SQUARED, 42
  - NUMERIC, 42, 43
  - NloptAlgorithms, 43
  - NumericalIntegrators, 43
  - QUADRATIC, 42
  - RUNGE\_KUTTA\_5TH, 43
  - RUNGE\_KUTTA\_CLASSIC, 43
  - SINGLE, 42
  - SLSQP, 43
  - teb::Config, 113
  - UPPER, 42
  - ZERO, 42
  - ZERO\_HESSIAN, 43
- teb::TebPlotter
  - EPS, 302
  - JPEG, 301
  - PDF, 302
  - PNG, 301
  - SVG, 301
  - TIKZ, 302
- teb::BaseController, 45
  - firstControl, 46
  - firstState, 46
  - getAbsoluteTimeVec, 46
  - getDt, 46
  - getN, 47
  - getStateCtrlInfoMat, 47
  - lastState, 47
  - resetController, 47
  - returnControlInputSequence, 47
  - step, 47
- teb::BaseEdge
  - ~BaseEdge, 52
  - \_hessians, 56
  - \_jacob\_backup, 56
  - \_jacobians, 57
  - \_values, 57
  - \_vertices, 57



- [\\_vertices\\_dim](#), 57
- [allocateMemory](#), 52
- [backupJacobian](#), 52
- [BaseEdge](#), 52
- [calculateVertexDimensions](#), 52
- [computeHessian](#), 53
- [computeJacobian](#), 53
- [computeValues](#), 54
- [Dimension](#), 57
- [dimension](#), 54
- [discardJacobianBackup](#), 54
- [EdgeContainer](#), 51
- [functType](#), 54
- [getCustomData](#), 54
- [getJacobianBackup](#), 54
- [getVertex](#), 54, 55
- [hessians](#), 55
- [isBoundConstraint](#), 55
- [jacobians](#), 55
- [NoVertices](#), 57
- [noVertices](#), 55
- [restoreJacobian](#), 55
- [ValueVector](#), 51
- [ValueVectorMap](#), 52
- [values](#), 56
- [valuesData](#), 56
- [valuesMap](#), 56
- [vertices](#), 56
- [teb::BaseEdge< D, FuncType >](#), 58
  - [~BaseEdge](#), 61
  - [\\_hessians](#), 66
  - [\\_jacob\\_backup](#), 66
  - [\\_jacobians](#), 66
  - [\\_values](#), 67
  - [\\_vertices](#), 67
  - [\\_vertices\\_dim](#), 67
  - [allocateMemory](#), 61
  - [backupJacobian](#), 61
  - [BaseEdge](#), 61
  - [calculateVertexDimensions](#), 62
  - [computeHessian](#), 62
  - [computeJacobian](#), 62
  - [computeValues](#), 63
  - [Dimension](#), 67
  - [dimension](#), 63
  - [discardJacobianBackup](#), 63
  - [EdgeContainer](#), 61
  - [functType](#), 63
  - [getCustomData](#), 63
  - [getJacobianBackup](#), 63
  - [getVertex](#), 64
  - [hessians](#), 64
  - [isBoundConstraint](#), 64
  - [jacobians](#), 64
  - [noVertices](#), 64
  - [resizeVertexContainer](#), 65
  - [restoreJacobian](#), 65
  - [setVertex](#), 65
  - [ValueVector](#), 61
  - [ValueVectorMap](#), 61
  - [values](#), 65
  - [valuesData](#), 65
  - [valuesMap](#), 66
  - [vertices](#), 66
- [teb::BaseEdge< D, FuncType, Vertices >](#), 49
- [teb::BaseSolver](#), 67
  - [~BaseSolver](#), 69
  - [\\_active\\_vertices](#), 73
  - [\\_equalities](#), 73
  - [\\_equalities\\_dim](#), 73
  - [\\_graph\\_structure\\_modified](#), 73
  - [\\_inequalities](#), 74
  - [\\_inequalities\\_dim](#), 74
  - [\\_no\\_vert\\_backups](#), 74
  - [\\_objective\\_dim](#), 74
  - [\\_objectives](#), 74
  - [\\_opt\\_vec\\_dim](#), 74
  - [applyIncrement](#), 70
  - [applyOptVec](#), 70
  - [backupVertices](#), 70
  - [BaseSolver](#), 69
  - [cfg](#), 75
  - [discardBackupVertices](#), 70
  - [EdgeContainer](#), 69
  - [getDimEqualities](#), 70
  - [getDimInequalities](#), 71
  - [getDimObjectives](#), 71
  - [getOptVecCopy](#), 71
  - [getOptVecDimension](#), 71
  - [initWorkspaces](#), 71
  - [restoreVertices](#), 72
  - [restoreVerticesButKeepBackup](#), 72
  - [setConfig](#), 72
  - [solve](#), 72
  - [solveImpl](#), 73
  - [VertexContainer](#), 69
- [teb::BaseSolverLeastSquares](#), 75
  - [\\_active\\_vertices](#), 84
  - [\\_equalities](#), 84
  - [\\_equalities\\_dim](#), 84
  - [\\_graph\\_structure\\_modified](#), 84
  - [\\_inequalities](#), 84
  - [\\_inequalities\\_dim](#), 85
  - [\\_no\\_vert\\_backups](#), 85
  - [\\_objective\\_dim](#), 85
  - [\\_objectives](#), 85
  - [\\_opt\\_vec\\_dim](#), 85
  - [\\_val\\_dim](#), 85
  - [\\_values](#), 86
  - [\\_weight\\_adapt\\_count](#), 86
  - [\\_weight\\_equalities](#), 86
  - [\\_weight\\_inequalities](#), 86
  - [adaptWeights](#), 78
  - [applyIncrement](#), 79
  - [applyOptVec](#), 79
  - [backupVertices](#), 79

- BaseSolverLeastSquares, 78
- buildValueVector, 80
- cfg, 86
- discardBackupVertices, 80
- EdgeContainer, 78
- getChi2, 80
- getDimEqualities, 81
- getDimInequalities, 81
- getDimObjectives, 81
- getOptVecCopy, 81
- getOptVecDimension, 81
- getValueDimension, 82
- initWorkspaces, 82
- restoreVertices, 82
- restoreVerticesButKeepBackup, 82
- setConfig, 83
- solve, 83
- solveImpl, 83
- VertexContainer, 78
- teb::BaseSolverNonlinearProgramDense, 87
  - \_active\_vertices, 97
  - \_equalities, 97
  - \_equalities\_dim, 97
  - \_equality\_jacobian, 97
  - \_equality\_values, 97
  - \_graph\_structure\_modified, 97
  - \_inequalities, 98
  - \_inequalities\_dim, 98
  - \_inequality\_jacobian, 98
  - \_inequality\_values, 98
  - \_lagrangian\_gradient, 98
  - \_lagrangian\_gradient\_backup, 98
  - \_lagrangian\_hessian, 98
  - \_multiplier\_eq, 99
  - \_multiplier\_ineq, 99
  - \_no\_vert\_backups, 99
  - \_objective\_dim, 99
  - \_objective\_gradient, 99
  - \_objective\_value, 99
  - \_objectives, 99
  - \_opt\_vec\_dim, 100
  - applyIncrement, 90
  - applyOptVec, 90
  - backupVertices, 91
  - BaseSolverNonlinearProgramDense, 90
  - buildEqualityConstraintJacobian, 91
  - buildEqualityConstraintValueVector, 91
  - buildInequalityConstraintJacobian, 91
  - buildInequalityConstraintValueVector, 91
  - buildObjectiveGradient, 91
  - buildObjectiveValue, 92
  - calculateLagrangianGradient, 92
  - calculateLagrangianHessian, 92
  - calculateLagrangianHessianFullBFGS, 92
  - calculateLagrangianHessianNumerically, 92
  - cfg, 100
  - discardBackupVertices, 92
  - EdgeContainer, 90
  - getDimEqualities, 93
  - getDimInequalities, 93
  - getDimObjectives, 93
  - getOptVecCopy, 93
  - getOptVecDimension, 93
  - initHessianBFGS, 94
  - initSolverWorkspace, 94
  - initWorkspaces, 94
  - initializeLagrangeMultiplier, 94
  - MatMapRowMajor, 90
  - restoreVertices, 94
  - restoreVerticesButKeepBackup, 95
  - setConfig, 95
  - solve, 95
  - solveImpl, 95
  - VertexContainer, 90
- teb::BoundConstraint
  - \_bounds, 110
  - \_hessians, 110
  - \_jacob\_backup, 110
  - \_jacobians, 111
  - \_values, 111
  - \_vertices, 111
  - \_vertices\_dim, 111
  - allocateMemory, 104
  - backupJacobian, 104
  - BoundConstraint, 104
  - BoundType, 111
  - calculateVertexDimensions, 104
  - computeHessian, 104
  - computeJacobian, 105
  - computeValues, 105
  - Dimension, 111
  - dimension, 105
  - discardJacobianBackup, 105
  - EdgeContainer, 103
  - funcType, 105
  - getBounds, 105
  - getCustomData, 106
  - getJacobianBackup, 106
  - getVertex, 106
  - hessians, 107
  - isBoundConstraint, 107
  - jacobians, 107
  - NoBounds, 112
  - NoVertices, 112
  - noVertices, 107
  - restoreJacobian, 107
  - setBounds, 107–109
  - ValueVector, 103
  - ValueVectorMap, 103
  - values, 109
  - valuesData, 110
  - valuesMap, 110
  - vertices, 110
- teb::BoundConstraint< Bound\_type, Vertex, Bound\_ - vars, Index >, 100
- teb::Config, 112

- optim, 113
  - teb, 113
  - util, 113
- teb::Config::Optim, 181
  - force\_rebuild\_optim\_graph, 181
  - solver, 181
- teb::Config::Optim::Solver, 200
  - lsq, 201
  - nlopt, 201
  - nonlin\_prog, 201
  - solver\_iter, 201
- teb::Config::Optim::Solver::Lsq, 175
  - soft\_constr\_epsilon, 175
  - weight\_adaptation\_factor, 175
  - weight\_equalities, 176
  - weight\_inequalities, 176
- teb::Config::Optim::Solver::Nlopt, 176
  - algorithm, 177
  - max\_optimization\_time, 177
  - stopping\_criteria\_ftol\_abs, 177
  - stopping\_criteria\_ftol\_rel, 177
  - stopping\_criteria\_xtol\_abs, 177
  - stopping\_criteria\_xtol\_rel, 177
  - tolerance\_equalities, 177
  - tolerance\_inequalities, 177
- teb::Config::Optim::Solver::NonlinearProgram, 177
  - hessian, 178
  - linesearch, 178
- teb::Config::Optim::Solver::NonlinearProgram::Hessian, 161
  - bfgs\_damped\_mode, 162
  - hessian\_init, 162
  - hessian\_init\_identity\_scale, 162
  - hessian\_method, 162
- teb::Config::Optim::Solver::NonlinearProgram::LineSearch, 174
  - alpha\_init, 174
  - beta, 174
  - sigma, 175
- teb::Config::Teb, 269
  - diff\_method, 270
  - dt\_hyst, 270
  - dt\_min, 270
  - dt\_ref, 270
  - goal\_dist\_force\_reinit, 270
  - n\_max, 270
  - n\_min, 270
  - n\_pre, 270
  - teb\_iter, 271
- teb::Config::Utilities, 316
- teb::ControlVertex
  - \_backup, 123
  - \_controls, 124
  - \_dimension, 124
  - \_fixed\_ctrls, 124
  - \_opt\_vec\_idx, 124
  - BackupStackType, 116
  - ControlVector, 116
  - ControlVertex, 117
  - controls, 117
  - DimControls, 124
  - Dimension, 124
  - dimension, 117
  - dimensionFree, 118
  - discardTop, 118
  - fixed\_ctrls, 118
  - FixedCtrls, 116
  - getData, 118
  - getDataFree, 118
  - getOptVecIdx, 118
  - isFixedAll, 119
  - isFixedAny, 119
  - isFixedComp, 119
  - operator<<, 123
  - operator=, 119
  - operator==, 123
  - plus, 119
  - plusFree, 120
  - pop, 120
  - push, 120
  - ScalarType, 116
  - setControls, 120
  - setFixedAll, 120
  - setFixedCtrl, 121
  - setFixedCtrls, 121
  - setFree, 121
  - setOptVecIdx, 121
  - stackSize, 123
  - top, 123
  - VertexContainer, 117
- teb::ControlVertex< q >, 113
- teb::CustomBoundData, 124
  - index, 125
  - lower, 125
  - no\_lower, 125
  - no\_upper, 125
  - upper, 125
- teb::EdgeMinimizeTime, 126
  - \_data, 133
  - \_hessians, 133
  - \_jacob\_backup, 133
  - \_jacobians, 133
  - \_values, 133
  - \_vertices, 133
  - \_vertices\_dim, 133
  - allocateMemory, 129
  - backupJacobian, 129
  - calculateVertexDimensions, 129
  - computeHessian, 129
  - computeJacobian, 129
  - computeValues, 130
  - Dimension, 133
  - dimension, 130
  - discardJacobianBackup, 130
  - EdgeContainer, 128
  - EdgeMinimizeTime, 129

- functType, [130](#)
- getCustomData, [130](#)
- getJacobianBackup, [130](#)
- getVertex, [130](#)
- hessians, [131](#)
- isBoundConstraint, [131](#)
- jacobians, [131](#)
- NoVertices, [133](#)
- noVertices, [131](#)
- restoreJacobian, [131](#)
- setData, [131](#)
- ValueVector, [128](#)
- ValueVectorMap, [128](#)
- values, [132](#)
- valuesData, [132](#)
- valuesMap, [132](#)
- vertices, [132](#)
- teb::EdgeQuadraticForm
  - \_Q, [143](#)
  - \_R, [143](#)
  - \_hessians, [142](#)
  - \_jacob\_backup, [142](#)
  - \_jacobians, [142](#)
  - \_ref\_state, [143](#)
  - \_values, [143](#)
  - \_vertices, [143](#)
  - \_vertices\_dim, [143](#)
  - allocateMemory, [138](#)
  - backupJacobian, [138](#)
  - calculateVertexDimensions, [138](#)
  - computeHessian, [138](#)
  - computeJacobian, [138](#)
  - computeValues, [139](#)
  - Dimension, [143](#)
  - dimension, [139](#)
  - discardJacobianBackup, [139](#)
  - EdgeContainer, [137](#)
  - EdgeQuadraticForm, [137](#)
  - functType, [139](#)
  - getCustomData, [139](#)
  - getJacobianBackup, [139](#)
  - getVertex, [140](#)
  - hessians, [140](#)
  - isBoundConstraint, [140](#)
  - jacobians, [140](#)
  - NoVertices, [143](#)
  - noVertices, [140](#)
  - restoreJacobian, [141](#)
  - setReference, [141](#)
  - setWeights, [141](#)
  - StateVector, [137](#)
  - ValueVector, [137](#)
  - ValueVectorMap, [137](#)
  - values, [141](#)
  - valuesData, [141](#), [142](#)
  - valuesMap, [142](#)
  - vertices, [142](#)
- teb::EdgeQuadraticForm< p, q >, [134](#)
- teb::EdgeSystemDynamics
  - \_hessians, [152](#)
  - \_jacob\_backup, [152](#)
  - \_jacobians, [152](#)
  - \_system, [152](#)
  - \_values, [152](#)
  - \_vertices, [152](#)
  - \_vertices\_dim, [153](#)
  - allocateMemory, [147](#)
  - backupJacobian, [148](#)
  - calculateVertexDimensions, [148](#)
  - computeHessian, [148](#)
  - computeJacobian, [148](#)
  - computeValues, [149](#)
  - Dimension, [153](#)
  - dimension, [149](#)
  - discardJacobianBackup, [149](#)
  - EdgeContainer, [147](#)
  - EdgeSystemDynamics, [147](#)
  - functType, [149](#)
  - getCustomData, [149](#)
  - getJacobianBackup, [149](#)
  - getVertex, [150](#)
  - hessians, [150](#)
  - isBoundConstraint, [150](#)
  - jacobians, [150](#)
  - NoVertices, [153](#)
  - noVertices, [150](#)
  - restoreJacobian, [151](#)
  - setSystemDynamics, [151](#)
  - SystemDynamics, [152](#)
  - ValueVector, [147](#)
  - ValueVectorMap, [147](#)
  - values, [151](#)
  - valuesData, [151](#)
  - valuesMap, [151](#)
  - vertices, [151](#)
- teb::EdgeSystemDynamics< p, q, central >, [144](#)
- teb::EdgeType, [153](#)
  - ~EdgeType, [155](#)
  - \_hessians, [158](#)
  - \_jacob\_backup, [158](#)
  - \_jacobians, [159](#)
  - allocateMemory, [156](#)
  - backupJacobian, [156](#)
  - computeHessian, [156](#)
  - computeJacobian, [156](#)
  - computeValues, [156](#)
  - dimension, [156](#)
  - discardJacobianBackup, [156](#)
  - EdgeContainer, [155](#)
  - EdgeType, [155](#)
  - functType, [156](#)
  - getCustomData, [157](#)
  - getJacobianBackup, [157](#)
  - getVertex, [157](#)
  - hessians, [157](#)
  - isBoundConstraint, [157](#)

- [jacobians](#), [157](#), [158](#)
  - [noVertices](#), [158](#)
  - [restoreJacobian](#), [158](#)
  - [ValueVectorMap](#), [155](#)
  - [valuesData](#), [158](#)
  - [valuesMap](#), [158](#)
- [teb::ExplicitEuler](#)
  - [~ExplicitEuler](#), [160](#)
  - [ControlVector](#), [160](#)
  - [ExplicitEuler](#), [160](#)
  - [integrate](#), [160](#)
  - [StateVector](#), [160](#)
- [teb::ExplicitEuler< p, q >](#), [159](#)
- [teb::HessianWorkspace](#), [162](#)
  - [~HessianWorkspace](#), [164](#)
  - [\\_workspace](#), [165](#)
  - [allocate](#), [164](#)
  - [getWorkspace](#), [164](#)
  - [HessianWorkspace](#), [164](#)
  - [WorkspaceMatrix](#), [164](#)
- [teb::HyperGraph](#), [165](#)
  - [\\_active\\_vertices](#), [171](#)
  - [\\_constraints\\_eq](#), [171](#)
  - [\\_constraints\\_ineq](#), [171](#)
  - [\\_graph\\_modified](#), [171](#)
  - [\\_objectives](#), [171](#)
  - [activeVertices](#), [167](#)
  - [addActiveVertex](#), [167](#)
  - [addEdgeEquality](#), [167](#)
  - [addEdgeInequality](#), [167](#)
  - [addEdgeObjective](#), [169](#)
  - [clearActiveVertices](#), [169](#)
  - [clearEdges](#), [169](#)
  - [clearGraph](#), [169](#)
  - [EdgeContainer](#), [167](#)
  - [equalities](#), [169](#)
  - [inequalities](#), [170](#)
  - [isGraphModified](#), [170](#)
  - [notifyGraphModified](#), [170](#)
  - [objectives](#), [170](#), [171](#)
  - [VertexContainer](#), [167](#)
- [teb::JacobianWorkspace](#), [172](#)
  - [~JacobianWorkspace](#), [173](#)
  - [\\_workspace](#), [174](#)
  - [allocate](#), [173](#)
  - [getWorkspace](#), [173](#)
  - [JacobianWorkspace](#), [173](#)
  - [WorkspaceMatrix](#), [173](#)
- [teb::NumericalIntegrator](#)
  - [~NumericalIntegrator](#), [180](#)
  - [ControlVector](#), [180](#)
  - [integrate](#), [180](#)
  - [NumericalIntegrator](#), [180](#)
  - [StateVector](#), [180](#)
- [teb::NumericalIntegrator< p, q >](#), [178](#)
- [teb::PlotOptions](#), [181](#)
  - [legend](#), [182](#)
  - [legend\\_entries](#), [182](#)
  - [skip\\_last\\_value\\_right\\_column](#), [182](#)
  - [title](#), [182](#)
  - [ylabels](#), [183](#)
- [teb::RungeKutta5thOrder](#)
  - [~RungeKutta5thOrder](#), [184](#)
  - [ControlVector](#), [184](#)
  - [integrate](#), [184](#)
  - [RungeKutta5thOrder](#), [184](#)
  - [StateVector](#), [184](#)
- [teb::RungeKutta5thOrder< p, q >](#), [183](#)
- [teb::RungeKuttaClassic](#)
  - [~RungeKuttaClassic](#), [186](#)
  - [ControlVector](#), [186](#)
  - [integrate](#), [187](#)
  - [RungeKuttaClassic](#), [186](#)
  - [StateVector](#), [186](#)
- [teb::RungeKuttaClassic< p, q >](#), [185](#)
- [teb::SimResults](#), [187](#)
  - [allocateMemory](#), [188](#)
  - [conservativeResize](#), [189](#)
  - [series](#), [189](#)
  - [Simulator](#), [189](#)
- [teb::SimResults::TimeSeries](#), [315](#)
  - [controls](#), [316](#)
  - [dt](#), [316](#)
  - [states](#), [316](#)
  - [time](#), [316](#)
- [teb::Simulator](#)
  - [~Simulator](#), [193](#)
  - [\\_callback\\_sim\\_pre](#), [199](#)
  - [\\_callback\\_step\\_post](#), [199](#)
  - [\\_callback\\_step\\_pre](#), [199](#)
  - [\\_control\\_bounds](#), [199](#)
  - [\\_controller](#), [199](#)
  - [\\_integrator](#), [199](#)
  - [\\_plotter](#), [200](#)
  - [\\_sample\\_time](#), [200](#)
  - [\\_system](#), [200](#)
  - [ControlVector](#), [192](#)
  - [IntegratorPtr](#), [192](#)
  - [plotResults](#), [193](#)
  - [sampleTime](#), [193](#)
  - [saturateControl](#), [194](#)
  - [setControlInputSaturation](#), [194](#)
  - [setIntegrator](#), [194](#)
  - [setPlotter](#), [194](#)
  - [setPostStepCallback](#), [195](#)
  - [setPreSimCallback](#), [195](#)
  - [setPreStepCallback](#), [195](#)
  - [setSampleTime](#), [196](#)
  - [simClosedLoop](#), [196](#), [197](#)
  - [simOpenAndClosedLoop](#), [197](#)
  - [simOpenLoop](#), [198](#)
  - [SimResults](#), [199](#)
  - [Simulator](#), [193](#)
  - [StateVector](#), [192](#)
  - [SystemDynamics](#), [199](#)
  - [systemStep](#), [198](#)

teb::Simulator< p, q >, 190  
 teb::SolverLevenbergMarquardtEigenDense, 201  
   \_active\_vertices, 210  
   \_equalities, 210  
   \_equalities\_dim, 211  
   \_graph\_structure\_modified, 211  
   \_inequalities, 211  
   \_inequalities\_dim, 211  
   \_jacobian, 211  
   \_no\_vert\_backups, 211  
   \_objective\_dim, 212  
   \_objectives, 212  
   \_opt\_vec\_dim, 212  
   \_val\_dim, 212  
   \_values, 212  
   \_weight\_adapt\_count, 212  
   \_weight\_equalities, 213  
   \_weight\_inequalities, 213  
   adaptWeights, 204  
   applyIncrement, 205  
   applyOptVec, 205  
   backupVertices, 205  
   buildJacobian, 206  
   buildValueVector, 206  
   cfg, 213  
   discardBackupVertices, 207  
   EdgeContainer, 204  
   getChi2, 207  
   getDimEqualities, 207  
   getDimInequalities, 207  
   getDimObjectives, 208  
   getOptVecCopy, 208  
   getOptVecDimension, 208  
   getValueDimension, 208  
   initWorkspaces, 209  
   restoreVertices, 209  
   restoreVerticesButKeepBackup, 209  
   setConfig, 209  
   solve, 209  
   solveImpl, 210  
   SolverLevenbergMarquardtEigenDense, 204  
   VertexContainer, 204  
 teb::SolverLevenbergMarquardtEigenSparse, 213  
   \_active\_vertices, 223  
   \_equalities, 223  
   \_equalities\_dim, 223  
   \_graph\_structure\_modified, 223  
   \_inequalities, 224  
   \_inequalities\_dim, 224  
   \_jacobian, 224  
   \_nnz\_per\_col, 224  
   \_no\_vert\_backups, 224  
   \_objective\_dim, 224  
   \_objectives, 224  
   \_opt\_vec\_dim, 225  
   \_sparse\_solver, 225  
   \_val\_dim, 225  
   \_values, 225  
   \_weight\_adapt\_count, 225  
   \_weight\_equalities, 225  
   \_weight\_inequalities, 225  
   adaptWeights, 217  
   allocateSparseJacobian, 217  
   applyIncrement, 217  
   applyOptVec, 218  
   backupVertices, 218  
   buildJacobian, 218  
   buildValueVector, 219  
   cfg, 226  
   countJacobianColNNZ, 219  
   discardBackupVertices, 219  
   EdgeContainer, 216  
   getChi2, 220  
   getDimEqualities, 220  
   getDimInequalities, 220  
   getDimObjectives, 220  
   getOptVecCopy, 220  
   getOptVecDimension, 221  
   getValueDimension, 221  
   initWorkspaces, 221  
   restoreVertices, 221  
   restoreVerticesButKeepBackup, 222  
   setConfig, 222  
   solve, 222  
   solveImpl, 222  
   SolverLevenbergMarquardtEigenSparse, 217  
   VertexContainer, 216  
 teb::SolverNloptPackage, 226  
   \_active\_vertices, 234  
   \_bound\_constraints, 234  
   \_bound\_constraints\_dim, 234  
   \_ctol\_eq, 234  
   \_ctol\_ineq, 234  
   \_equalities, 234  
   \_equalities\_dim, 234  
   \_graph\_structure\_modified, 235  
   \_inequalities, 235  
   \_inequalities\_dim, 235  
   \_inequalities\_without\_bounds, 235  
   \_inequalities\_without\_bounds\_dim, 235  
   \_lower\_bounds, 235  
   \_no\_vert\_backups, 235  
   \_objective\_dim, 236  
   \_objectives, 236  
   \_opt\_vec\_dim, 236  
   \_upper\_bounds, 236  
   applyIncrement, 229  
   applyOptVec, 230  
   backupVertices, 230  
   cfg, 236  
   constraintsEq, 230  
   constraintsInEq, 230  
   discardBackupVertices, 230  
   EdgeContainer, 229  
   equalityConstraintFunction, 230  
   getBoundConstrDimFromStorage, 231

- getDimEqualities, 231
- getDimInequalities, 231
- getDimObjectives, 231
- getEqConstrDimFromStorage, 231
- getInEqConstrDimFromStorage, 231
- getOptVecCopy, 231
- getOptVecDimFromStorage, 232
- getOptVecDimension, 232
- inequalityConstraintFunction, 232
- initWorkspaces, 232
- objectiveFunction, 232
- objectives, 232
- restoreVertices, 232
- restoreVerticesButKeepBackup, 233
- seperateInequalitiesAndBounds, 233
- setConfig, 233
- solve, 233
- solveImpl, 233
- SolverNloptPackage, 229
- VertexContainer, 229
- teb::SolverSQPDense, 237
  - \_A, 250
  - \_active\_vertices, 250
  - \_delta, 250
  - \_dmultiplier\_eq, 250
  - \_dmultiplier\_ineq, 250
  - \_equalities, 251
  - \_equalities\_dim, 251
  - \_equality\_jacobian, 251
  - \_equality\_values, 251
  - \_graph\_structure\_modified, 251
  - \_increment, 251
  - \_inequalities, 252
  - \_inequalities\_dim, 252
  - \_inequality\_jacobian, 252
  - \_inequality\_values, 252
  - \_lagrangian\_gradient, 252
  - \_lagrangian\_gradient\_backup, 252
  - \_lagrangian\_hessian, 253
  - \_lb, 253
  - \_merit\_alpha, 253
  - \_merit\_grad, 253
  - \_multiplier\_eq, 253
  - \_multiplier\_ineq, 253
  - \_no\_vert\_backups, 253
  - \_objective\_dim, 253
  - \_objective\_gradient, 254
  - \_objective\_value, 254
  - \_objectives, 254
  - \_opt\_vec\_dim, 254
  - \_q dual, 254
  - \_qsolver, 254
  - \_ub, 254
  - applyIncrement, 241
  - applyOptVec, 242
  - backupVertices, 242
  - buildEqualityConstraintJacobian, 242
  - buildEqualityConstraintValueVector, 242
  - buildInequalityConstraintJacobian, 242
  - buildInequalityConstraintValueVector, 243
  - buildObjectiveGradient, 243
  - buildObjectiveValue, 243
  - calculateLagrangianGradient, 243
  - calculateLagrangianHessian, 243
  - calculateLagrangianHessianFullBFGS, 244
  - calculateLagrangianHessianNumerically, 244
  - calculateMerit, 244
  - calculateMeritDerivative, 244
  - cfg, 255
  - checkConvergence, 245
  - discardBackupVertices, 245
  - EdgeContainer, 240
  - getDimEqualities, 245
  - getDimInequalities, 245
  - getDimObjectives, 245
  - getOptVecCopy, 245
  - getOptVecDimension, 246
  - initHessianBFGS, 246
  - initSolverWorkspace, 246
  - initWorkspaces, 247
  - initializeLagrangeMultiplier, 246
  - MatMapRowMajor, 240
  - restoreVertices, 247
  - restoreVerticesButKeepBackup, 247
  - setConfig, 247
  - solve, 248
  - solveImpl, 248
  - SolverSQPDense, 240
  - VertexContainer, 240
- teb::StateVertex
  - \_backup, 265
  - \_dimension, 265
  - \_fixed\_states, 265
  - \_opt\_vec\_idx, 265
  - \_states, 265
  - BackupStackType, 258
  - DimStates, 265
  - Dimension, 265
  - dimension, 259
  - dimensionFree, 259
  - discardTop, 259
  - fixed\_states, 259
  - FixedStates, 258
  - getData, 260
  - getDataFree, 260
  - getOptVecIdx, 260
  - isFixedAll, 260
  - isFixedAny, 260
  - isFixedComp, 260
  - operator<<, 264
  - operator=, 261
  - operator==, 264
  - plus, 261
  - plusFree, 261
  - pop, 261
  - push, 261



- ScalarType, 258
- setFixedAll, 262
- setFixedState, 262
- setFixedStates, 262
- setFree, 263
- setOptVecIdx, 263
- setStates, 263
- stackSize, 263
- StateVector, 258
- StateVertex, 259
- states, 263, 264
- top, 264
- VertexContainer, 258
- teb::StateVertex< p >, 255
- teb::SystemDynamics
  - ~SystemDynamics, 267
  - ControlVector, 267
  - DimControls, 269
  - DimStates, 269
  - EdgeSystemDynamics, 269
  - finiteElemCentralDiff, 267
  - finiteElemFwdEuler, 268
  - Simulator, 269
  - stateSpaceModel, 268
  - StateVector, 267
  - SystemDynamics, 267
- teb::SystemDynamics< p, q >, 265
- teb::TebController
  - ~TebController, 277
  - \_active\_control\_bounds, 296
  - \_active\_quadratic\_form, 297
  - \_active\_state\_bounds, 297
  - \_active\_system\_dynamics, 297
  - \_active\_time\_optimal, 297
  - \_cfg\_owned, 298
  - \_ctrl\_seq, 298
  - \_dt, 298
  - \_dt\_ref, 298
  - \_fixed\_goal\_states, 298
  - \_goal\_backup, 298
  - \_graph, 298
  - \_no\_samples, 299
  - \_optimized, 299
  - \_solver, 299
  - \_state\_seq, 299
  - activateControlBounds, 277, 278
  - activateObjectiveQuadraticForm, 278
  - activateObjectiveTimeOptimal, 279
  - activateStateBounds, 279, 280
  - buildOptimizationGraph, 280
  - cfg, 299
  - ControlSequence, 276
  - controlSequence, 281
  - ControlVector, 276
  - customOptimizationGraph, 281
  - customOptimizationGraphHotStart, 281
  - dt, 282
  - EdgeContainer, 276
  - firstControl, 282
  - firstControlRef, 282
  - firstControlSaturated, 282
  - firstState, 283
  - firstStateRef, 283
  - FixedStates, 276
  - getAbsoluteTimeVec, 283
  - getActiveVertices, 283
  - getDt, 284
  - getM, 284
  - getN, 284
  - getSampleOptVecIdxVMat, 284
  - getStateCtrlInfoMat, 285
  - getTEBFixedMap, 285
  - graph, 285
  - initOptimization, 286
  - initTrajectory, 286
  - insertStateControlInputPair, 286
  - lastState, 287
  - lastStateRef, 287
  - NoControls, 299
  - NoStates, 299
  - optimizeTEB, 287
  - predictControl, 287
  - pushBackStateControlInputPair, 288
  - pushFrontStateControlInputPair, 288
  - removeStateControlInputPair, 288
  - resampleTrajectory, 289
  - resetController, 289
  - resizeTrajectory, 289
  - returnControlInputSequence, 290
  - setFixedDt, 290
  - setFixedGoal, 290, 291
  - setFixedStart, 291
  - setGoalStatesFixedOrUnfixed, 291, 293
  - setSolver, 293
  - setSystemDynamics, 294
  - setupHorizon, 294
  - StateSequence, 276
  - stateSequence, 295
  - StateVector, 276
  - step, 295
  - TebController, 277
  - updateGoal, 295
  - updateStart, 296
  - VertexContainer, 277
- teb::TebController< p, q >, 271
- teb::TebPlotter, 300
  - ~TebPlotter, 302
  - \_file\_export, 307
  - \_pdf\_flag, 307
  - clearWindow, 302
  - closeWindow, 302
  - completePdfExport, 302
  - FileFormat, 301
  - isExportToFileEnabled, 302
  - pipe, 307
  - plot, 303



- plot1DVector, 303
- plotCustomKey, 303
- plotMulti, 304
- plotTEB, 304, 305
- plotTwoCol, 305
- setOutputToFile, 306
- setOutputToWindow, 306
- spyMatrix, 306
- switchWindow, 306
- TebPlotter, 302
- teb::TimeDiff, 307
  - ~TimeDiff, 311
  - \_backup, 314
  - \_dt, 314
  - \_fixed, 315
  - \_opt\_vec\_idx, 315
  - Dimension, 315
  - dimension, 311
  - dimensionFree, 311
  - discardTop, 311
  - dt, 311
  - getData, 311
  - getDataFree, 311
  - getOptVecIdx, 312
  - isFixedAll, 312
  - isFixedAny, 312
  - isFixedComp, 312
  - operator<<, 314
  - operator(), 312
  - operator=, 312
  - plus, 313
  - plusFree, 313
  - pop, 313
  - push, 313
  - setFixedAll, 313
  - setFree, 313
  - setOptVecIdx, 314
  - stackSize, 314
  - TimeDiff, 310
  - top, 314
  - VertexContainer, 310
- teb::VertexType, 316
  - \_opt\_vec\_idx, 321
  - dimension, 318
  - dimensionFree, 318
  - discardTop, 318
  - getData, 319
  - getDataFree, 319
  - getOptVecIdx, 319
  - isFixedAll, 319
  - isFixedAny, 319
  - isFixedComp, 319
  - plus, 320
  - plusFree, 320
  - pop, 320
  - push, 320
  - setFree, 320
  - setOptVecIdx, 320
  - stackSize, 320
  - top, 320
  - VertexContainer, 318
- teb\_controller.h, 336
- teb\_controller.hpp, 336
- teb\_iter
  - teb::Config::Teb, 271
- teb\_plotter.cpp, 336
- teb\_plotter.h, 337
- teb\_plotter.hpp, 337
- teb\_vertices.h, 337
- TebController
  - teb::TebController, 277
- TebPlotter
  - teb::TebPlotter, 302
- time
  - teb::SimResults::TimeSeries, 316
- TimeDiff
  - teb::TimeDiff, 310
- title
  - teb::PlotOptions, 182
- tolerance\_equalities
  - teb::Config::Optim::Solver::Nlopt, 177
- tolerance\_inequalities
  - teb::Config::Optim::Solver::Nlopt, 177
- top
  - teb::ControlVertex, 123
  - teb::StateVertex, 264
  - teb::TimeDiff, 314
  - teb::VertexType, 320
- typedefs.h, 338
  - INF, 339
  - INPUT\_STREAM, 339
  - PI, 339
  - PRINT\_DEBUG, 339
  - PRINT\_DEBUG\_COND, 339
  - PRINT\_DEBUG\_ONCE, 339
  - PRINT\_ERROR, 340
  - PRINT\_INFO, 340
  - PRINT\_INFO\_COND, 340
  - PRINT\_INFO\_ONCE, 340
- UPPER
  - teb, 42
- updateGoal
  - teb::TebController, 295
- updateStart
  - teb::TebController, 296
- upper
  - teb::CustomBoundData, 125
- util
  - teb::Config, 113
- utilities.h, 340
- ValueVector
  - teb::BaseEdge, 51
  - teb::BaseEdge< D, FuncType >, 61
  - teb::BoundConstraint, 103
  - teb::EdgeMinimizeTime, 128

- teb::EdgeQuadraticForm, [137](#)
- teb::EdgeSystemDynamics, [147](#)
- ValueVectorMap
  - teb::BaseEdge, [52](#)
  - teb::BaseEdge< D, FuncType >, [61](#)
  - teb::BoundConstraint, [103](#)
  - teb::EdgeMinimizeTime, [128](#)
  - teb::EdgeQuadraticForm, [137](#)
  - teb::EdgeSystemDynamics, [147](#)
  - teb::EdgeType, [155](#)
- values
  - teb::BaseEdge, [56](#)
  - teb::BaseEdge< D, FuncType >, [65](#)
  - teb::BoundConstraint, [109](#)
  - teb::EdgeMinimizeTime, [132](#)
  - teb::EdgeQuadraticForm, [141](#)
  - teb::EdgeSystemDynamics, [151](#)
- valuesData
  - teb::BaseEdge, [56](#)
  - teb::BaseEdge< D, FuncType >, [65](#)
  - teb::BoundConstraint, [110](#)
  - teb::EdgeMinimizeTime, [132](#)
  - teb::EdgeQuadraticForm, [141](#), [142](#)
  - teb::EdgeSystemDynamics, [151](#)
  - teb::EdgeType, [158](#)
- valuesMap
  - teb::BaseEdge, [56](#)
  - teb::BaseEdge< D, FuncType >, [66](#)
  - teb::BoundConstraint, [110](#)
  - teb::EdgeMinimizeTime, [132](#)
  - teb::EdgeQuadraticForm, [142](#)
  - teb::EdgeSystemDynamics, [151](#)
  - teb::EdgeType, [158](#)
- VertexContainer
  - teb::BaseSolver, [69](#)
  - teb::BaseSolverLeastSquares, [78](#)
  - teb::BaseSolverNonlinearProgramDense, [90](#)
  - teb::ControlVertex, [117](#)
  - teb::HyperGraph, [167](#)
  - teb::SolverLevenbergMarquardtEigenDense, [204](#)
  - teb::SolverLevenbergMarquardtEigenSparse, [216](#)
  - teb::SolverNloptPackage, [229](#)
  - teb::SolverSQPDense, [240](#)
  - teb::StateVertex, [258](#)
  - teb::TebController, [277](#)
  - teb::TimeDiff, [310](#)
  - teb::VertexType, [318](#)
- vertices
  - teb::BaseEdge, [56](#)
  - teb::BaseEdge< D, FuncType >, [66](#)
  - teb::BoundConstraint, [110](#)
  - teb::EdgeMinimizeTime, [132](#)
  - teb::EdgeQuadraticForm, [142](#)
  - teb::EdgeSystemDynamics, [151](#)
- Visualization, [38](#)
- weight\_adaptation\_factor
  - teb::Config::Optim::Solver::Lsq, [175](#)
- weight\_equalities
  - teb::Config::Optim::Solver::Lsq, [176](#)
- weight\_inequalities
  - teb::Config::Optim::Solver::Lsq, [176](#)
- WorkspaceMatrix
  - teb::HessianWorkspace, [164](#)
  - teb::JacobianWorkspace, [173](#)
- workspaces.h, [341](#)
- ylabels
  - teb::PlotOptions, [183](#)
- ZERO
  - teb, [42](#)
- ZERO\_HESSIAN
  - teb, [43](#)