ABIP-LP

2.0.0

Generated by Doxygen 1.9.5

Chapter 1

README

This AMD and LDL code are from the official SuiteSparse library: a suite of sparse matrix algorithms authored or co-authored by Tim Davis, Texas A&M University. You can visit this repo by: https://github.com/DrTimothyAldenDavis/SuiteSparse/tree/dev. We include each license n the subfolder, respectively.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 ABIP_A_DATA_MATRIX Struct Reference

```
#include <amatrix.h>
```

Data Fields

- abip_float * x
- abip_int * i
- abip_int * p
- abip_int m
- abip_int n

4.1.1 Detailed Description

Definition at line 10 of file amatrix.h.

4.1.2 Field Documentation

4.1.2.1 i

```
abip_int* i
```

Definition at line 13 of file amatrix.h.

4.1.2.2 m

```
abip_int m
```

Definition at line 15 of file amatrix.h.

4.1.2.3 n

```
abip_int n
```

Definition at line 16 of file amatrix.h.

4.1.2.4 p

```
abip_int* p
```

Definition at line 14 of file amatrix.h.

4.1.2.5 x

```
abip_float* x
```

Definition at line 12 of file amatrix.h.

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

- linsys/amatrix.h

4.2 ABIP_ADAPTIVE_WORK Struct Reference

Data Fields

- abip_float * u_prev
- abip_float * v_prev
- abip_float * ut
- abip_float * u
- abip_float * v
- abip_float * ut_next
- abip_float * u_next
- abip_float * v_next
- abip_float * delta_ut
- abip_float * delta_u
- abip_float * delta_v
- abip_int l
- abip_int k
- abip_float total_adapt_time

4.2.1 Detailed Description

Definition at line 13 of file adaptive.c.

4.2.2 Field Documentation

4.2.2.1 delta_u

```
abip_float* delta_u
```

Definition at line 25 of file adaptive.c.

4.2.2.2 delta_ut

```
abip_float* delta_ut
```

Definition at line 24 of file adaptive.c.

4.2.2.3 delta_v

```
abip_float* delta_v
```

Definition at line 26 of file adaptive.c.

4.2.2.4 k

```
abip_int k
```

Definition at line 29 of file adaptive.c.

4.2.2.5 l

```
abip_int l
```

Definition at line 28 of file adaptive.c.

4.2.2.6 total_adapt_time

`abip_float total_adapt_time`

Definition at line 31 of file adaptive.c.

4.2.2.7 u

`abip_float* u`

Definition at line 18 of file adaptive.c.

4.2.2.8 u_next

`abip_float* u_next`

Definition at line 21 of file adaptive.c.

4.2.2.9 u_prev

`abip_float* u_prev`

Definition at line 15 of file adaptive.c.

4.2.2.10 ut

`abip_float* ut`

Definition at line 17 of file adaptive.c.

4.2.2.11 ut_next

`abip_float* ut_next`

Definition at line 20 of file adaptive.c.

4.2.2.12 v

```
abip_float* v
```

Definition at line 19 of file adaptive.c.

4.2.2.13 v_next

```
abip_float* v_next
```

Definition at line 22 of file adaptive.c.

4.2.2.14 v_prev

```
abip_float* v_prev
```

Definition at line 16 of file adaptive.c.

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

- src/adaptive.c

4.3 ABIP_INFO Struct Reference

```
#include <abip.h>
```

Data Fields

- char status [32]
- abip_int status_val
- abip_int ipm_iter
- abip_int admm_iter
- abip_float pobj
- abip_float dobj
- abip_float res_pri
- abip_float res_dual
- abip_float rel_gap
- abip_float res_infeas
- abip_float res_unbdd
- abip_float setup_time
- abip_float solve_time

4.3.1 Detailed Description

Definition at line 88 of file abip.h.

4.3.2 Field Documentation

4.3.2.1 admm_iter

`abip_int admm_iter`

Definition at line 93 of file abip.h.

4.3.2.2 dobj

`abip_float dobj`

Definition at line 96 of file abip.h.

4.3.2.3 ipm_iter

`abip_int ipm_iter`

Definition at line 92 of file abip.h.

4.3.2.4 pobj

`abip_float pobj`

Definition at line 95 of file abip.h.

4.3.2.5 rel_gap

`abip_float rel_gap`

Definition at line 99 of file abip.h.

4.3.2.6 res_dual

```
abip_float res_dual
```

Definition at line 98 of file abip.h.

4.3.2.7 res_infeas

```
abip_float res_infeas
```

Definition at line 100 of file abip.h.

4.3.2.8 res_pri

```
abip_float res_pri
```

Definition at line 97 of file abip.h.

4.3.2.9 res_unbdd

```
abip_float res_unbdd
```

Definition at line 101 of file abip.h.

4.3.2.10 setup_time

```
abip_float setup_time
```

Definition at line 103 of file abip.h.

4.3.2.11 solve_time

```
abip_float solve_time
```

Definition at line 104 of file abip.h.

4.3.2.12 status

```
char status[32]
```

Definition at line 90 of file abip.h.

4.3.2.13 status_val

```
abip_int status_val
```

Definition at line 91 of file abip.h.

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

- include/abip.h

4.4 ABIP_LIN_SYS_WORK Struct Reference

```
#include <direct.h>
```

Data Fields

- cs ∗ L
- abip_float ∗ D
- abip_int ∗ P
- abip_int ∗ i
- abip_int ∗ j
- abip_float ∗ bp
- void ∗ pardiso_work [PARDISOINDEX]
- abip_float total_solve_time
- abip_float ∗ p
- abip_float ∗ r
- abip_float ∗ Gp
- abip_float ∗ tmp
- ABIPMatrix ∗ At
- abip_float ∗ z
- abip_float ∗ M
- abip_int tot_cg_its

4.4.1 Detailed Description

Definition at line 16 of file direct.h.

4.4.2 Field Documentation

4.4.2.1 At

```
ABIPMatrix* At
```

Definition at line 20 of file indirect.h.

4.4.2.2 bp

```
abip_float* bp
```

Definition at line 23 of file direct.h.

4.4.2.3 D

```
abip_float* D
```

Definition at line 19 of file direct.h.

4.4.2.4 Gp

```
abip_float* Gp
```

Definition at line 18 of file indirect.h.

4.4.2.5 i

```
abip_int* i
```

Definition at line 21 of file direct.h.

4.4.2.6 j

```
abip_int* j
```

Definition at line 22 of file direct.h.

4.4.2.7 L

```
cs* L
```

Definition at line 18 of file direct.h.

4.4.2.8 M

```
abip_float* M
```

Definition at line 24 of file indirect.h.

4.4.2.9 P

```
abip_int* P
```

Definition at line 20 of file direct.h.

4.4.2.10 p

```
abip_float* p
```

Definition at line 16 of file indirect.h.

4.4.2.11 pardiso_work

```
void* pardiso_work[PARDISOINDEX]
```

Definition at line 25 of file direct.h.

4.4.2.12 r

```
abip_float* r
```

Definition at line 17 of file indirect.h.

4.4.2.13 tmp

```
abip_float* tmp
```

Definition at line 19 of file indirect.h.

4.4.2.14 tot_cg_its

```
abip_int tot_cg_its
```

Definition at line 27 of file indirect.h.

4.4.2.15 total_solve_time

```
abip_float total_solve_time
```

Definition at line 26 of file direct.h.

4.4.2.16 z

```
abip_float* z
```

Definition at line 23 of file indirect.h.

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

- linsys/direct.h
- linsys/indirect.h

4.5 ABIP_PROBLEM_DATA Struct Reference

```
#include <abip.h>
```

Data Fields

- abip_int m
- abip_int n
- ABIPMatrix * A
- abip_float * b
- abip_float * c
- abip_float sp
- ABIPSettings * stgs

4.5.1 Detailed Description

Definition at line 23 of file abip.h.

4.5.2 Field Documentation

4.5.2.1 A

```
ABIPMatrix* A
```

Definition at line 27 of file abip.h.

4.5.2.2 b

```
abip_float* b
```

Definition at line 29 of file abip.h.

4.5.2.3 c

```
abip_float* c
```

Definition at line 30 of file abip.h.

4.5.2.4 m

```
abip_int m
```

Definition at line 25 of file abip.h.

4.5.2.5 n

```
abip_int n
```

Definition at line 26 of file abip.h.

4.5.2.6 sp

```
abip_float sp
```

Definition at line 31 of file abip.h.

4.5.2.7 stgs

```
ABIPSettings* stgs
```

Definition at line 33 of file abip.h.

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

- include/abip.h

4.6 ABIP_RESIDUALS Struct Reference

```
#include <abip.h>
```

Data Fields

- abip_int last_ipm_iter
- abip_int last_admm_iter
- abip_float last_mu
- abip_float res_pri
- abip_float res_dual
- abip_float rel_gap
- abip_float res_infeas
- abip_float res_unbdd
- abip_float ct_x_by_tau
- abip_float bt_y_by_tau
- abip_float tau
- abip_float kap

4.6.1 Detailed Description

Definition at line 178 of file abip.h.

4.6.2 Field Documentation

4.6.2.1 bt_y_by_tau

```
abip_float bt_y_by_tau
```

Definition at line 191 of file abip.h.

4.6.2.2 ct_x_by_tau

```
abip_float ct_x_by_tau
```

Definition at line 190 of file abip.h.

4.6.2.3 kap

```
abip_float kap
```

Definition at line 194 of file abip.h.

4.6.2.4 last_admm_iter

```
abip_int last_admm_iter
```

Definition at line 181 of file abip.h.

4.6.2.5 last_ipm_iter

```
abip_int last_ipm_iter
```

Definition at line 180 of file abip.h.

4.6.2.6 last_mu

```
abip_float last_mu
```

Definition at line 182 of file abip.h.

4.6.2.7 rel_gap

```
abip_float rel_gap
```

Definition at line 186 of file abip.h.

4.6.2.8 res_dual

```
abip_float res_dual
```

Definition at line 185 of file abip.h.

4.6.2.9 res_infeas

```
abip_float res_infeas
```

Definition at line 187 of file abip.h.

4.6.2.10 res_pri

```
abip_float res_pri
```

Definition at line 184 of file abip.h.

4.6.2.11 res_unbdd

```
abip_float res_unbdd
```

Definition at line 188 of file abip.h.

4.6.2.12 tau

```
abip_float tau
```

Definition at line 193 of file abip.h.

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

- include/abip.h

4.7 ABIP_SCALING Struct Reference

```
#include <abip.h>
```

Data Fields

- abip_float * D
- abip_float * E
- abip_float mean_norm_row_A
- abip_float mean_norm_col_A

4.7.1 Detailed Description

Definition at line 107 of file abip.h.

4.7.2 Field Documentation

4.7.2.1 D

```
abip_float* D
```

Definition at line 109 of file abip.h.

4.7.2.2 E

```
abip_float* E
```

Definition at line 110 of file abip.h.

4.7.2.3 mean_norm_col_A

```
abip_float mean_norm_col_A
```

Definition at line 113 of file abip.h.

4.7.2.4 mean_norm_row_A

```
abip_float mean_norm_row_A
```

Definition at line 112 of file abip.h.

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

- include/abip.h

4.8 ABIP_SETTINGS Struct Reference

```
#include <abip.h>
```

Data Fields

- abip_int normalize
- abip_int pfeasopt
- abip_float scale
- abip_float rho_y
- abip_float sparsity_ratio
- abip_int max_ipm_iters
- abip_int max_admm_iters
- abip_float max_time
- abip_float eps
- abip_float alpha
- abip_float cg_rate
- abip_int adaptive
- abip_float eps_cor
- abip_float eps_pen
- abip_float dynamic_sigma
- abip_float dynamic_x
- abip_float dynamic_eta
- abip_int restart_fre
- abip_int restart_thresh
- abip_int verbose
- abip_int warm_start
- abip_int adaptive_lookback
- abip_int origin_rescale
- abip_int pc_ruiz_rescale
- abip_int qp_rescale
- abip_int ruiz_iter
- abip_int hybrid_mu
- abip_float hybrid_thresh
- abip_float dynamic_sigma_second
- abip_int half_update
- abip_int avg_criterion

4.8.1 Detailed Description

Definition at line 36 of file abip.h.

4.8.2 Field Documentation

4.8.2.1 adaptive

`abip_int adaptive`

Definition at line 52 of file abip.h.

4.8.2.2 adaptive_lookback

`abip_int adaptive_lookback`

Definition at line 67 of file abip.h.

4.8.2.3 alpha

`abip_float alpha`

Definition at line 49 of file abip.h.

4.8.2.4 avg_criterion

`abip_int avg_criterion`

Definition at line 78 of file abip.h.

4.8.2.5 cg_rate

`abip_float cg_rate`

Definition at line 50 of file abip.h.

4.8.2.6 dynamic_eta

```
abip_float dynamic_eta
```

Definition at line 59 of file abip.h.

4.8.2.7 dynamic_sigma

```
abip_float dynamic_sigma
```

Definition at line 56 of file abip.h.

4.8.2.8 dynamic_sigma_second

```
abip_float dynamic_sigma_second
```

Definition at line 76 of file abip.h.

4.8.2.9 dynamic_x

```
abip_float dynamic_x
```

Definition at line 58 of file abip.h.

4.8.2.10 eps

```
abip_float eps
```

Definition at line 48 of file abip.h.

4.8.2.11 eps_cor

```
abip_float eps_cor
```

Definition at line 53 of file abip.h.

4.8.2.12 eps_pen

```
abip_float eps_pen
```

Definition at line 54 of file abip.h.

4.8.2.13 half_update

```
abip_int half_update
```

Definition at line 77 of file abip.h.

4.8.2.14 hybrid_mu

```
abip_int hybrid_mu
```

Definition at line 74 of file abip.h.

4.8.2.15 hybrid_thresh

```
abip_float hybrid_thresh
```

Definition at line 75 of file abip.h.

4.8.2.16 max_admm_iters

```
abip_int max_admm_iters
```

Definition at line 45 of file abip.h.

4.8.2.17 max_ipm_iters

```
abip_int max_ipm_iters
```

Definition at line 44 of file abip.h.

4.8.2.18 max_time

`abip_float max_time`

Definition at line 46 of file abip.h.

4.8.2.19 normalize

`abip_int normalize`

Definition at line 38 of file abip.h.

4.8.2.20 origin_rescale

`abip_int origin_rescale`

Definition at line 70 of file abip.h.

4.8.2.21 pc_ruiz_rescale

`abip_int pc_ruiz_rescale`

Definition at line 71 of file abip.h.

4.8.2.22 pfeasopt

`abip_int pfeasopt`

Definition at line 39 of file abip.h.

4.8.2.23 qp_rescale

`abip_int qp_rescale`

Definition at line 72 of file abip.h.

4.8.2.24 restart_fre

```
abip_int restart_fre
```

Definition at line 61 of file abip.h.

4.8.2.25 restart_thresh

```
abip_int restart_thresh
```

Definition at line 62 of file abip.h.

4.8.2.26 rho_y

```
abip_float rho_y
```

Definition at line 41 of file abip.h.

4.8.2.27 ruiz_iter

```
abip_int ruiz_iter
```

Definition at line 73 of file abip.h.

4.8.2.28 scale

```
abip_float scale
```

Definition at line 40 of file abip.h.

4.8.2.29 sparsity_ratio

```
abip_float sparsity_ratio
```

Definition at line 42 of file abip.h.

4.8.2.30 verbose

```
abip_int verbose
```

Definition at line 64 of file abip.h.

4.8.2.31 warm_start

```
abip_int warm_start
```

Definition at line 65 of file abip.h.

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

- include/abip.h

4.9 ABIP_SOL_VARS Struct Reference

```
#include <abip.h>
```

Data Fields

- abip_float * x
- abip_float * y
- abip_float * s

4.9.1 Detailed Description

Definition at line 81 of file abip.h.

4.9.2 Field Documentation

4.9.2.1 s

```
abip_float* s
```

Definition at line 85 of file abip.h.

4.9.2.2 x

`abip_float* x`

Definition at line 83 of file abip.h.

4.9.2.3 y

`abip_float* y`

Definition at line 84 of file abip.h.

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

- include/abip.h

4.10 ABIP_WORK Struct Reference

`#include <abip.h>`

Data Fields

- abip_float sigma
- abip_float gamma
- abip_int final_check
- abip_int double_check
- abip_float mu
- abip_float beta
- abip_float ∗ u
- abip_float ∗ v
- abip_float ∗ u_t
- abip_float ∗ u_prev
- abip_float ∗ v_prev
- abip_float ∗ u_avg
- abip_float ∗ v_avg
- abip_float ∗ u_avgcon
- abip_float ∗ v_avgcon
- abip_float ∗ u_sumcon
- abip_float ∗ v_sumcon
- abip_int fre_old
- abip_float ∗ h
- abip_float ∗ g
- abip_float ∗ pr
- abip_float ∗ dr
- abip_float g_th
- abip_float sc_b
- abip_float sc_c
- abip_float nm_b

- abip_float nm_c
- abip_float ∗ b
- abip_float ∗ c
- abip_int m
- abip_int n
- ABIPMatrix ∗ A
- abip_float sp
- ABIPLinSysWork ∗ p
- ABIPAdaptWork ∗ adapt
- ABIPSettings ∗ stgs
- ABIPScaling ∗ scal

4.10.1 Detailed Description

Definition at line 126 of file abip.h.

4.10.2 Field Documentation

4.10.2.1 A

```
ABIPMatrix* A
```

Definition at line 169 of file abip.h.

4.10.2.2 adapt

```
ABIPAdaptWork* adapt
```

Definition at line 173 of file abip.h.

4.10.2.3 b

```
abip_float* b
```

Definition at line 165 of file abip.h.

4.10.2.4 beta

`abip_float beta`

Definition at line 134 of file abip.h.

4.10.2.5 c

`abip_float* c`

Definition at line 166 of file abip.h.

4.10.2.6 double_check

`abip_int double_check`

Definition at line 131 of file abip.h.

4.10.2.7 dr

`abip_float* dr`

Definition at line 157 of file abip.h.

4.10.2.8 final_check

`abip_int final_check`

Definition at line 130 of file abip.h.

4.10.2.9 fre_old

`abip_int fre_old`

Definition at line 152 of file abip.h.

4.10.2.10 g

```
abip_float* g
```

Definition at line 155 of file abip.h.

4.10.2.11 g_th

```
abip_float g_th
```

Definition at line 159 of file abip.h.

4.10.2.12 gamma

```
abip_float gamma
```

Definition at line 129 of file abip.h.

4.10.2.13 h

```
abip_float* h
```

Definition at line 154 of file abip.h.

4.10.2.14 m

```
abip_int m
```

Definition at line 167 of file abip.h.

4.10.2.15 mu

```
abip_float mu
```

Definition at line 133 of file abip.h.

4.10.2.16 n

```
abip_int n
```

Definition at line 168 of file abip.h.

4.10.2.17 nm_b

```
abip_float nm_b
```

Definition at line 162 of file abip.h.

4.10.2.18 nm_c

```
abip_float nm_c
```

Definition at line 163 of file abip.h.

4.10.2.19 p

```
ABIPLinSysWork* p
```

Definition at line 172 of file abip.h.

4.10.2.20 pr

```
abip_float* pr
```

Definition at line 156 of file abip.h.

4.10.2.21 sc_b

```
abip_float sc_b
```

Definition at line 160 of file abip.h.

4.10.2.22 sc_c

```
abip_float sc_c
```

Definition at line 161 of file abip.h.

4.10.2.23 scal

```
ABIPScaling* scal
```

Definition at line 175 of file abip.h.

4.10.2.24 sigma

```
abip_float sigma
```

Definition at line 128 of file abip.h.

4.10.2.25 sp

```
abip_float sp
```

Definition at line 170 of file abip.h.

4.10.2.26 stgs

```
ABIPSettings* stgs
```

Definition at line 174 of file abip.h.

4.10.2.27 u

```
abip_float* u
```

Definition at line 136 of file abip.h.

4.10.2.28 u_avg

```
abip_float* u_avg
```

Definition at line 142 of file abip.h.

4.10.2.29 u_avgcon

```
abip_float* u_avgcon
```

Definition at line 145 of file abip.h.

4.10.2.30 u_prev

```
abip_float* u_prev
```

Definition at line 139 of file abip.h.

4.10.2.31 u_sumcon

```
abip_float* u_sumcon
```

Definition at line 148 of file abip.h.

4.10.2.32 u_t

```
abip_float* u_t
```

Definition at line 138 of file abip.h.

4.10.2.33 v

```
abip_float* v
```

Definition at line 137 of file abip.h.

4.10.2.34 v_avg

```
abip_float* v_avg
```

Definition at line 143 of file abip.h.

4.10.2.35 v_avgcon

```
abip_float* v_avgcon
```

Definition at line 146 of file abip.h.

4.10.2.36 v_prev

```
abip_float* v_prev
```

Definition at line 140 of file abip.h.

4.10.2.37 v_sumcon

```
abip_float* v_sumcon
```

Definition at line 149 of file abip.h.

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

- include/abip.h

4.11 SuiteSparse_config_struct Struct Reference

```
#include <SuiteSparse_config.h>
```

Data Fields

- void ∗(∗ malloc_func)(size_t)
- void ∗(∗ calloc_func)(size_t, size_t)
- void ∗(∗ realloc_func)(void ∗, size_t)
- void(∗ free_func)(void ∗)
- int(∗ printf_func)(const char ∗,...)
- abip_float(∗ hypot_func)(abip_float, abip_float)
- int(∗ divcomplex_func)(abip_float, abip_float, abip_float, abip_float, abip_float ∗, abip_float ∗)

4.11.1 Detailed Description

Definition at line 87 of file SuiteSparse_config.h.

4.11.2 Field Documentation

4.11.2.1 calloc_func

```
void *(* calloc_func) (size_t, size_t)
```

Definition at line 90 of file SuiteSparse_config.h.

4.11.2.2 divcomplex_func

```
int(* divcomplex_func) (abip_float, abip_float, abip_float, abip_float, abip_float *, abip_float
*)
```

Definition at line 95 of file SuiteSparse_config.h.

4.11.2.3 free_func

```
void(* free_func) (void *)
```

Definition at line 92 of file SuiteSparse_config.h.

4.11.2.4 hypot_func

```
abip_float(* hypot_func) (abip_float, abip_float)
```

Definition at line 94 of file SuiteSparse_config.h.

4.11.2.5 malloc_func

```
void *(* malloc_func) (size_t)
```

Definition at line 89 of file SuiteSparse_config.h.

4.11.2.6 printf_func

```
int(* printf_func) (const char *,...)
```

Definition at line 93 of file SuiteSparse_config.h.

4.11.2.7 realloc_func

```
void *(* realloc_func) (void *, size_t)
```

Definition at line 91 of file SuiteSparse_config.h.

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

- external/SuiteSparse_config.h

Chapter 5

File Documentation

5.1 compile_direct.m File Reference

Functions

- function compile_direct (flags, common_abip) abip_include
- if exist (lib_path) % % platform
- fprintf ("Unknown platform. Not using MKL \n")
- end End if if (flags.COMPILE_WITH_OPENMP) cmd = sprintf('-DEXTRA_VERBOSE %s', flags.LCFLAG)
- eval (cmd)

Variables

- platform = convertCharsToStrings(computer('arch'))
- lib_path = ""
- If use MKL
- If use we suggest you set the environmental variables by oneapi
- If use we suggest you set the environmental variables by it is typically placed at opt intel oneapi setvars sh alternatively
- If use we suggest you set the environmental variables by it is typically placed at opt intel oneapi setvars sh you can set the lib_path to your MKL path by your self
- If use we suggest you set the environmental variables by it is typically placed at opt intel oneapi setvars sh you can set the lib_path to your MKL path by your For example
- If use we suggest you set the environmental variables by it is typically placed at opt intel oneapi setvars sh you can set the lib_path to your MKL path by your For in linux
- If use we suggest you set the environmental variables by it is typically placed at opt intel oneapi setvars sh you can set the lib_path to your MKL path by your For in you may find it at opt intel oneapi mkl lib intel64
- If use we suggest you set the environmental variables by it is typically placed at opt intel oneapi setvars sh you can set the lib_path to your MKL path by your For in you may find it at opt intel oneapi mkl lib then you uncomment the following
- end mkl_macro = "-DABIP_PARDISO"
- pardiso_src = fullfile("linsys", "abip_pardiso.c")
- flags link
- if mexext
- else cmd = sprintf ('mex -O %s %s %s %s %s %s %s %s', mkl_macro, flags.arr, lib_path, flags.LCFLAG, flags.INCS, abip_include, flags.INT)
- end ldl_path = fullfile("external", "ldl")
- amd_path = fullfile("external", "amd")
- ldl_files = ["ldl.c"]
- amd_files
- abip_ldl = fullfile(ldl_path, ldl_files)
- abip_amd = fullfile(amd_path, amd_files)

5.1.1 Function Documentation

5.1.1.1 compile_direct()

```
function compile_direct (
            flags ,
            common_abip )
```

5.1.1.2 eval()

```
eval (
            cmd )
```

5.1.1.3 exist()

```
if exist (
            lib_path )
```

5.1.1.4 fprintf()

```
fprintf (
            "Unknown platform.  Not using MKL \n" )
```

5.1.1.5 if()

```
end if (
            flags.  COMPILE_WITH_OPENMP ) = sprintf('-DEXTRA_VERBOSE %s', flags.LCFLAG)
```

5.1.2 Variable Documentation

5.1.2.1 abip_amd

```
abip_amd = fullfile(amd_path, amd_files)
```

Definition at line 73 of file compile_direct.m.

5.1.2.2 abip_ldl

```
abip_ldl = fullfile(ldl_path, ldl_files)
```

Definition at line 71 of file compile_direct.m.

5.1.2.3 alternatively

```
If use we suggest you set the environmental variables by it is typically placed at opt intel
oneapi setvars sh alternatively
```

Definition at line 14 of file compile_direct.m.

5.1.2.4 amd_files

```
amd_files
```

Initial value:

```
= ["amd_order", "amd_dump", "amd_postorder", "amd_post_tree", ...
    "amd_aat", "amd_2", "amd_1", "amd_defaults", "amd_control", ...
    "amd_info", "amd_valid", "amd_global", "amd_preprocess"]
```

Definition at line 66 of file compile_direct.m.

5.1.2.5 amd_path

```
amd_path = fullfile("external", "amd")
```

Definition at line 63 of file compile_direct.m.

5.1.2.6 cmd

```
cmd = sprintf ('mex -O %s %s %s %s %s %s %s %s', mkl_macro, flags.arr, lib_path, flags.LCFLAG,
flags.INCS, abip_include, flags.INT)
```

Definition at line 59 of file compile_direct.m.

5.1.2.7 example

```
If use we suggest you set the environmental variables by it is typically placed at opt intel
oneapi setvars sh you can set the lib_path to your MKL path by your For example
```

Definition at line 15 of file compile_direct.m.

5.1.2.8 following

```
If use we suggest you set the environmental variables by it is typically placed at opt intel
oneapi setvars sh you can set the lib_path to your MKL path by your For in you may find it at
opt intel oneapi mkl lib then you uncomment the following
```

Definition at line 16 of file compile_direct.m.

5.1.2.9 intel64

```
If use we suggest you set the environmental variables by it is typically placed at opt intel
oneapi setvars sh you can set the lib_path to your MKL path by your For in you may find it at
opt intel oneapi mkl lib intel64
```

Definition at line 15 of file compile_direct.m.

5.1.2.10 ldl_files

```
ldl_files = ["ldl.c"]
```

Definition at line 65 of file compile_direct.m.

5.1.2.11 ldl_path

```
end ldl_path = fullfile("external", "ldl")
```

Definition at line 62 of file compile_direct.m.

5.1.2.12 lib_path

```
else lib_path = ""
```

Definition at line 10 of file compile_direct.m.

5.1.2.13 link

```
else flags link
```

Initial value:

```
= [flags.link, ' -lmkl_intel_ilp64',...
        ' -lmkl_core', ' -lmkl_sequential ']
```

Definition at line 34 of file compile_direct.m.

5.1.2.14 linux

```
If use we suggest you set the environmental variables by it is typically placed at opt intel
oneapi setvars sh you can set the lib_path to your MKL path by your For in linux
```

Definition at line 15 of file compile_direct.m.

5.1.2.15 mexext

```
if mexext
```

Initial value:

```
== "mexw64"
        flags.link = '-lut -lmwblas -lmwlapack'
```

Definition at line 47 of file compile_direct.m.

5.1.2.16 MKL

```
If use MKL
```

Definition at line 12 of file compile_direct.m.

5.1.2.17 mkl_macro

```
mkl_macro = "-DABIP_PARDISO"
```

Definition at line 29 of file compile_direct.m.

5.1.2.18 oneapi

```
If use we suggest you set the environmental variables by oneapi
```

Definition at line 12 of file compile_direct.m.

5.1.2.19 pardiso_src

```
end End if pardiso_src = fullfile("linsys", "abip_pardiso.c")
```

Definition at line 30 of file compile_direct.m.

5.1.2.20 platform

```
elseif platform = convertCharsToStrings(computer('arch'))
```

Definition at line 9 of file compile_direct.m.

5.1.2.21 self

```
If use we suggest you set the environmental variables by it is typically placed at opt intel
oneapi setvars sh you can set the lib_path to your MKL path by your self
```

Definition at line 14 of file compile_direct.m.

5.2 compile_direct.m

Go to the documentation of this file.

```
function compile_direct(flags, common_abip)

abip_include = strjoin("-I" + [fullfile("include");
    fullfile("linsys");
    fullfile("external");
    fullfile("external", "amd");
    fullfile("external", "ldl")]);

platform = convertCharsToStrings(computer('arch'));
lib_path = "";
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If use MKL, we suggest you set the environmental variables by oneapi,
%   it is typically placed at /opt/intel/oneapi/setvars.sh
%   alternatively, you can set the lib_path to your MKL path by your self,
% For example, in linux, you may find it at /opt/intel/oneapi/mkl/2021.2.0/lib/intel64,
%   then you uncomment the following:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lib_path = "/opt/intel/oneapi/mkl/2021.2.0/lib/";
% if exist(lib_path) % #ok
%     platform = computer('arch');
%     lib_path = "C:\'Program Files (x86)'\Intel\oneAPI\mkl\2022.1.0\lib\intel64";
%     lib_path = "-L" + lib_path;
% else
%     platform = "nomkl";
% end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mkl_macro = "-DABIP_PARDISO";
pardiso_src = fullfile("linsys", "abip_pardiso.c");

if platform == "win64"
    fprintf("Linking MKL in Windows \n");
    flags.link = [flags.link, ' -lmkl_intel_ilp64',...
        ' -lmkl_core', ' -lmkl_sequential '];
elseif platform == "maci64"
    fprintf("Linking MKL in MacOS \n");
    flags.link = [flags.link, ' -lmkl_intel_ilp64',...
        ' -lmkl_core', ' -lmkl_sequential '];
elseif platform == "glnxa64"
    fprintf("Linking MKL in Linux \n");
    flags.link = [flags.link, ' -lmkl_intel_ilp64',...
        ' -lmkl_core', ' -lmkl_sequential '];
else
    lib_path = '';
    mkl_macro = '';
    if mexext == "mexw64"
        flags.link = '-lut -lmwblas -lmwlapack';
    else
        flags.link = '-lm -lut -lmwblas -lmwlapack';
    end % End if
    pardiso_src = '';
    fprintf("Unknown platform. Not using MKL \n");
end % End if

if (flags.COMPILE_WITH_OPENMP)
    cmd = sprintf('mex -O %s %s %s %s COMPFLAGS="/openmp \\$COMPFLAGS" CFLAGS="\\$CFLAGS -fopenmp" %s
 %s', flags.arr, flags.LCFLAG, flags.INCS, abip_include, flags.INT);
else
    cmd = sprintf ('mex -O %s %s %s %s %s %s %s %s', mkl_macro, flags.arr, lib_path, flags.LCFLAG,
 flags.INCS, abip_include, flags.INT);
end

ldl_path = fullfile("external", "ldl");
amd_path = fullfile("external", "amd");

ldl_files = ["ldl.c"];
amd_files = ["amd_order", "amd_dump", "amd_postorder", "amd_post_tree", ...
    "amd_aat", "amd_2", "amd_1", "amd_defaults", "amd_control", ...
    "amd_info", "amd_valid", "amd_global", "amd_preprocess"];
amd_files = amd_files + ".c";

abip_ldl = fullfile(ldl_path, ldl_files);
abip_ldl = strjoin(abip_ldl);
abip_amd = fullfile(amd_path, amd_files);
abip_amd = strjoin(abip_amd);

cmd = sprintf("%s %s %s %s %s", cmd, abip_amd, abip_ldl,...
    fullfile("linsys", "direct.c"), pardiso_src);
cmd = sprintf ('%s %s %s %s %s -output abip_direct', cmd, common_abip, flags.link, flags.LOCS,
 flags.BLASLIB);

```

```
00080 eval(cmd);
00081
```

5.3 compile_indirect.m File Reference

Functions

- function compile_indirect (flags, common_abip) abip_include
- compile indirect if (flags.COMPILE_WITH_OPENMP) cmd
- end eval (cmd)

Variables

- else cmd

5.3.1 Function Documentation

5.3.1.1 compile_indirect()

```
function compile_indirect (
            flags ,
            common_abip )
```

5.3.1.2 eval()

```
end eval (
            cmd )
```

5.3.1.3 if()

```
compile indirect if (
            flags. COMPILE_WITH_OPENMP )
```

5.3.2 Variable Documentation

5.3.2.1 cmd

```
else cmd
```

Initial value:

```
= sprintf('mex -O %s %s %s %s %s %s %s %s %s %s -output abip_indirect',...
        flags.arr, flags.LCFLAG, common_abip, flags.INCS,...
        fullfile("linsys", "indirect.c"),...
        flags.link, abip_include, flags.LOCS, flags.BLASLIB, flags.INT)
```

Definition at line 13 of file compile_indirect.m.

5.4 compile_indirect.m

Go to the documentation of this file.

```
00001 function compile_indirect(flags, common_abip)
00002
00003 abip_include = strjoin("-I" + [fullfile("include");
00004     fullfile("linsys")]);
00005
00006 % compile indirect
00007 if (flags.COMPILE_WITH_OPENMP)
00008     cmd = sprintf('mex -O %s %s %s %s COMPFLAGS="/openmp \\$COMPFLAGS" CFLAGS="\\$CFLAGS -fopenmp" %s
       %s %s %s %s %s -output abip_indirect',...
00009         flags.arr, flags.LCFLAG, common_abip, flags.INCS,...
00010         fullfile("linsys", "direct", "private.c"),...
00011         flags.link, abip_include, flags.LOCS, flags.BLASLIB, flags.INT);
00012 else
00013     cmd = sprintf('mex -O %s %s %s %s %s %s %s %s %s %s -output abip_indirect',...
00014         flags.arr, flags.LCFLAG, common_abip, flags.INCS,...
00015         fullfile("linsys", "indirect.c"),...
00016         flags.link, abip_include, flags.LOCS, flags.BLASLIB, flags.INT);
00017 end
00018
00019 eval(cmd);
```

5.5 external/amd/amd.h File Reference

```
#include <stddef.h>
#include "SuiteSparse_config.h"
```

Macros

- #define EXTERN extern
- #define AMD_CONTROL 5 /∗ size of Control array ∗/
- #define AMD_INFO 20 /∗ size of Info array ∗/
- #define AMD_DENSE 0 /∗ "dense" if degree > Control [0] ∗ sqrt (n) ∗/
- #define AMD_AGGRESSIVE 1 /∗ do aggressive absorption if Control [1] != 0 ∗/
- #define AMD_DEFAULT_DENSE 10.0 /∗ default "dense" degree 10∗sqrt(n) ∗/
- #define AMD_DEFAULT_AGGRESSIVE 1 /∗ do aggressive absorption by default ∗/
- #define AMD_STATUS 0 /∗ return value of amd_order and amd_l_order ∗/
- #define AMD_N 1 /∗ A is n-by-n ∗/
- #define AMD_NZ 2 /∗ number of nonzeros in A ∗/
- #define AMD_SYMMETRY 3 /∗ symmetry of pattern (1 is sym., 0 is unsym.) ∗/
- #define AMD_NZDIAG 4 /∗ # of entries on diagonal ∗/
- #define AMD_NZ_A_PLUS_AT 5 /∗ nz in A+A' ∗/
- #define AMD_NDENSE 6 /∗ number of "dense" rows/columns in A ∗/

- #define AMD_MEMORY 7 /* amount of memory used by AMD */
- #define AMD_NCMPA 8 /* number of garbage collections in AMD */
- #define AMD_LNZ 9 /* approx. nz in L, excluding the diagonal */
- #define AMD_NDIV 10 /* number of fl. point divides for LU and LDL' */
- #define AMD_NMULTSUBS_LDL 11 /* number of fl. point (*,-) pairs for LDL' */
- #define AMD_NMULTSUBS_LU 12 /* number of fl. point (*,-) pairs for LU */
- #define AMD_DMAX 13 /* max nz. in any column of L, incl. diagonal */
- #define AMD_OK 0 /* success */
- #define AMD_OUT_OF_MEMORY -1 /* malloc failed, or problem too large */
- #define AMD_INVALID -2 /* input arguments are not valid */
- #define AMD_OK_BUT_JUMBLED
- #define AMD_DATE "May 4, 2016"
- #define AMD_VERSION_CODE(main, sub) ((main) * 1000 + (sub))
- #define AMD_MAIN_VERSION 2
- #define AMD_SUB_VERSION 4
- #define AMD_SUBSUB_VERSION 6
- #define AMD_VERSION AMD_VERSION_CODE(AMD_MAIN_VERSION,AMD_SUB_VERSION)

Functions

- int amd_order (int n, const int Ap[ ], const int Ai[ ], int P[ ], abip_float Control[ ], abip_float ABIPInfo[ ])
- SuiteSparse_long amd_l_order (SuiteSparse_long n, const SuiteSparse_long Ap[ ], const SuiteSparse_long Ai[ ], SuiteSparse_long P[ ], abip_float Control[ ], abip_float ABIPInfo[ ])
- void amd_2 (int n, int Pe[ ], int Iw[ ], int Len[ ], int iwlen, int pfree, int Nv[ ], int Next[ ], int Last[ ], int Head[ ], int Elen[ ], int Degree[ ], int W[ ], abip_float Control[ ], abip_float ABIPInfo[ ])
- void amd_l2 (SuiteSparse_long n, SuiteSparse_long Pe[ ], SuiteSparse_long Iw[ ], SuiteSparse_long Len[ ], SuiteSparse_long iwlen, SuiteSparse_long pfree, SuiteSparse_long Nv[ ], SuiteSparse_long Next[ ], SuiteSparse_long Last[ ], SuiteSparse_long Head[ ], SuiteSparse_long Elen[ ], SuiteSparse_long Degree[ ], SuiteSparse_long W[ ], abip_float Control[ ], abip_float ABIPInfo[ ])
- int amd_valid (int n_row, int n_col, const int Ap[ ], const int Ai[ ])
- SuiteSparse_long amd_l_valid (SuiteSparse_long n_row, SuiteSparse_long n_col, const SuiteSparse_long Ap[ ], const SuiteSparse_long Ai[ ])
- void amd_defaults (abip_float Control[ ])
- void amd_l_defaults (abip_float Control[ ])
- void amd_control (abip_float Control[ ])
- void amd_l_control (abip_float Control[ ])
- void amd_info (abip_float ABIPInfo[ ])
- void amd_l_info (abip_float ABIPInfo[ ])

Variables

- EXTERN void *(* amd_malloc)(size_t)
- EXTERN void(* amd_free)(void *)
- EXTERN void *(* amd_realloc)(void *, size_t)
- EXTERN void *(* amd_calloc)(size_t, size_t)
- EXTERN int(* amd_printf)(const char *,...)

5.5.1 Macro Definition Documentation

5.5.1.1 AMD_AGGRESSIVE

```
#define AMD_AGGRESSIVE 1 /* do aggressive absorption if Control [1] != 0 */
```

Definition at line 341 of file amd.h.

5.5.1.2 AMD_CONTROL

```
#define AMD_CONTROL 5 /* size of Control array */
```

Definition at line 336 of file amd.h.

5.5.1.3 AMD_DATE

```
#define AMD_DATE "May 4, 2016"
```

Definition at line 394 of file amd.h.

5.5.1.4 AMD_DEFAULT_AGGRESSIVE

```
#define AMD_DEFAULT_AGGRESSIVE 1 /* do aggressive absorption by default */
```

Definition at line 345 of file amd.h.

5.5.1.5 AMD_DEFAULT_DENSE

```
#define AMD_DEFAULT_DENSE 10.0 /* default "dense" degree 10*sqrt(n) */
```

Definition at line 344 of file amd.h.

5.5.1.6 AMD_DENSE

```
#define AMD_DENSE 0 /* "dense" if degree > Control [0] * sqrt (n) */
```

Definition at line 340 of file amd.h.

5.5.1.7 AMD_DMAX

```
#define AMD_DMAX 13 /* max nz.  in any column of L, incl.  diagonal */
```

Definition at line 361 of file amd.h.

5.5.1.8 AMD_INFO

```
#define AMD_INFO 20 /* size of Info array */
```

Definition at line 337 of file amd.h.

5.5.1.9 AMD_INVALID

```
#define AMD_INVALID -2 /* input arguments are not valid */
```

Definition at line 369 of file amd.h.

5.5.1.10 AMD_LNZ

```
#define AMD_LNZ 9 /* approx.  nz in L, excluding the diagonal */
```

Definition at line 357 of file amd.h.

5.5.1.11 AMD_MAIN_VERSION

```
#define AMD_MAIN_VERSION 2
```

Definition at line 396 of file amd.h.

5.5.1.12 AMD_MEMORY

```
#define AMD_MEMORY 7 /* amount of memory used by AMD */
```

Definition at line 355 of file amd.h.

5.5.1.13 AMD_N

```
#define AMD_N 1 /* A is n-by-n */
```

Definition at line 349 of file amd.h.

5.5.1.14 AMD_NCMPA

```
#define AMD_NCMPA 8 /* number of garbage collections in AMD */
```

Definition at line 356 of file amd.h.

5.5.1.15 AMD_NDENSE

```
#define AMD_NDENSE 6 /* number of "dense" rows/columns in A */
```

Definition at line 354 of file amd.h.

5.5.1.16 AMD_NDIV

```
#define AMD_NDIV 10 /* number of fl.  point divides for LU and LDL' */
```

Definition at line 358 of file amd.h.

5.5.1.17 AMD_NMULTSUBS_LDL

```
#define AMD_NMULTSUBS_LDL 11 /* number of fl.  point (*,-) pairs for LDL' */
```

Definition at line 359 of file amd.h.

5.5.1.18 AMD_NMULTSUBS_LU

```
#define AMD_NMULTSUBS_LU 12 /* number of fl.  point (*,-) pairs for LU */
```

Definition at line 360 of file amd.h.

5.5.1.19 AMD_NZ

```
#define AMD_NZ 2 /* number of nonzeros in A */
```

Definition at line 350 of file amd.h.

5.5.1.20 AMD_NZ_A_PLUS_AT

```
#define AMD_NZ_A_PLUS_AT 5 /* nz in A+A' */
```

Definition at line 353 of file amd.h.

5.5.1.21 AMD_NZDIAG

```
#define AMD_NZDIAG 4 /* # of entries on diagonal */
```

Definition at line 352 of file amd.h.

5.5.1.22 AMD_OK

```
#define AMD_OK 0 /* success */
```

Definition at line 367 of file amd.h.

5.5.1.23 AMD_OK_BUT_JUMBLED

```
#define AMD_OK_BUT_JUMBLED
```

Value:

```
1        /* input matrix is OK for amd_order, but
* columns were not sorted, and/or duplicate entries were present.  AMD had
* to do extra work before ordering the matrix.  This is a warning, not an
* error.  */
```

Definition at line 370 of file amd.h.

5.5.1.24 AMD_OUT_OF_MEMORY

```
#define AMD_OUT_OF_MEMORY -1 /* malloc failed, or problem too large */
```

Definition at line 368 of file amd.h.

5.5.1.25 AMD_STATUS

```
#define AMD_STATUS 0 /* return value of amd_order and amd_l_order */
```

Definition at line 348 of file amd.h.

5.5.1.26 AMD_SUB_VERSION

```
#define AMD_SUB_VERSION 4
```

Definition at line 397 of file amd.h.

5.5.1.27 AMD_SUBSUB_VERSION

```
#define AMD_SUBSUB_VERSION 6
```

Definition at line 398 of file amd.h.

5.5.1.28 AMD_SYMMETRY

```
#define AMD_SYMMETRY 3 /* symmetry of pattern (1 is sym., 0 is unsym.)  */
```

Definition at line 351 of file amd.h.

5.5.1.29 AMD_VERSION

```
#define AMD_VERSION AMD_VERSION_CODE(AMD_MAIN_VERSION,AMD_SUB_VERSION)
```

Definition at line 399 of file amd.h.

5.5.1.30 AMD_VERSION_CODE

```
#define AMD_VERSION_CODE(
              main,
              sub ) ((main) * 1000 + (sub))
```

Definition at line 395 of file amd.h.

5.5.1.31 EXTERN

```
#define EXTERN extern
```

Definition at line 311 of file amd.h.

5.5.2 Function Documentation

5.5.2.1 amd_2()

```
void amd_2 (
            int n,
            int Pe[ ],
            int Iw[ ],
            int Len[ ],
            int iwlen,
            int pfree,
            int Nv[ ],
            int Next[ ],
            int Last[ ],
            int Head[ ],
            int Elen[ ],
            int Degree[ ],
            int W[ ],
            abip_float Control[ ],
            abip_float ABIPInfo[ ] )
```

5.5.2.2 amd_control()

```
void amd_control (
            abip_float Control[ ] )
```

5.5.2.3 amd_defaults()

```
void amd_defaults (
            abip_float Control[ ] )
```

5.5.2.4 amd_info()

```
void amd_info (
            abip_float ABIPInfo[ ] )
```

5.5.2.5 amd_l2()

```
void amd_l2 (
            SuiteSparse_long n,
            SuiteSparse_long Pe[ ],
            SuiteSparse_long Iw[ ],
            SuiteSparse_long Len[ ],
            SuiteSparse_long iwlen,
            SuiteSparse_long pfree,
            SuiteSparse_long Nv[ ],
            SuiteSparse_long Next[ ],
            SuiteSparse_long Last[ ],
            SuiteSparse_long Head[ ],
            SuiteSparse_long Elen[ ],
            SuiteSparse_long Degree[ ],
            SuiteSparse_long W[ ],
            abip_float Control[ ],
            abip_float ABIPInfo[ ] )
```

5.5.2.6 amd_l_control()

```
void amd_l_control (
            abip_float Control[ ] )
```

5.5.2.7 amd_l_defaults()

```
void amd_l_defaults (
            abip_float Control[ ] )
```

5.5.2.8 amd_l_info()

```
void amd_l_info (
            abip_float ABIPInfo[ ] )
```

5.5.2.9 amd_l_order()

```
SuiteSparse_long amd_l_order (
            SuiteSparse_long n,
            const SuiteSparse_long Ap[ ],
            const SuiteSparse_long Ai[ ],
            SuiteSparse_long P[ ],
            abip_float Control[ ],
            abip_float ABIPInfo[ ] )
```

5.5.2.10 amd_l_valid()

```
SuiteSparse_long amd_l_valid (
            SuiteSparse_long n_row,
            SuiteSparse_long n_col,
            const SuiteSparse_long Ap[],
            const SuiteSparse_long Ai[] )
```

5.5.2.11 amd_order()

```
int amd_order (
            int n,
            const int Ap[],
            const int Ai[],
            int P[],
            abip_float Control[],
            abip_float ABIPInfo[] )
```

5.5.2.12 amd_valid()

```
int amd_valid (
            int n_row,
            int n_col,
            const int Ap[],
            const int Ai[] )
```

5.5.3 Variable Documentation

5.5.3.1 amd_calloc

```
EXTERN void *(* amd_calloc) (size_t, size_t) (
            size_t ,
            size_t  )
```

Definition at line 317 of file amd.h.

5.5.3.2 amd_free

```
EXTERN void(* amd_free) (void *) (
            void *  )
```

Definition at line 315 of file amd.h.

5.5.3.3 amd_malloc

```
EXTERN void *(* amd_malloc) (size_t) (
            size_t )
```

Definition at line 314 of file amd.h.

5.5.3.4 amd_printf

```
EXTERN int(* amd_printf) (const char *,...) (
            const char * ,
             ... )
```

Definition at line 318 of file amd.h.

5.5.3.5 amd_realloc

```
EXTERN void *(* amd_realloc) (void *, size_t) (
            void * ,
            size_t )
```

Definition at line 316 of file amd.h.

5.6 amd.h

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD:  approximate minimum degree ordering =========================== */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD Version 2.4, Copyright (c) 1996-2013 by Timothy A. Davis,             */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* AMD finds a symmetric ordering P of a matrix A so that the Cholesky
 * factorization of P*A*P' has fewer nonzeros and takes less work than the
 * Cholesky factorization of A.  If A is not symmetric, then it performs its
 * ordering on the matrix A+A'.  Two sets of user-callable routines are
 * provided, one for int integers and the other for SuiteSparse_long integers.
 *
 * The method is based on the approximate minimum degree algorithm, discussed
 * in Amestoy, Davis, and Duff, "An approximate degree ordering algorithm",
 * SIAM Journal of Matrix Analysis and Applications, vol. 17, no. 4, pp.
 * 886-905, 1996.  This package can perform both the AMD ordering (with
 * aggressive absorption), and the AMDBAR ordering (without aggressive
 * absorption) discussed in the above paper.  This package differs from the
 * Fortran codes discussed in the paper:
 *
 *       (1) it can ignore "dense" rows and columns, leading to faster run times
 *       (2) it computes the ordering of A+A' if A is not symmetric
 *       (3) it is followed by a depth-first post-ordering of the assembly tree
 *           (or supernodal elimination tree)
 *
 * For historical reasons, the Fortran versions, amd.f and amdbar.f, have
 * been left (nearly) unchanged.  They compute the identical ordering as
 * described in the above paper.
 */
```

```c

#ifndef AMD_H
#define AMD_H

/* make it easy for C++ programs to include AMD */
#ifdef __cplusplus
extern "C" {
#endif

/* get the definition of size_t: */
#include <stddef.h>

#include "SuiteSparse_config.h"

int amd_order                          /* returns AMD_OK, AMD_OK_BUT_JUMBLED,
                                        * AMD_INVALID, or AMD_OUT_OF_MEMORY */
(
            int n,      /* A is n-by-n.  n must be >= 0. */
            const int Ap [ ],      /* column pointers for A, of size n+1 */
            const int Ai [ ],      /* row indices of A, of size nz = Ap [n] */
            int P [ ],      /* output permutation, of size n */
            abip_float Control [ ],      /* input Control settings, of size AMD_CONTROL */
            abip_float ABIPInfo [ ]      /* output ABIPInfo statistics, of size AMD_INFO */
) ;

SuiteSparse_long amd_l_order    /* see above for description of arguments */
(
            SuiteSparse_long n,
            const SuiteSparse_long Ap [ ],
            const SuiteSparse_long Ai [ ],
            SuiteSparse_long P [ ],
            abip_float Control [ ],
            abip_float ABIPInfo [ ]
) ;

/* Input arguments (not modified):
 *
 *       n: the matrix A is n-by-n.
 *       Ap: an int/SuiteSparse_long array of size n+1, containing column
 *              pointers of A.
 *       Ai: an int/SuiteSparse_long array of size nz, containing the row
 *              indices of A, where nz = Ap [n].
 *       Control:  a double array of size AMD_CONTROL, containing control
 *           parameters.  Defaults are used if Control is ABIP_NULL.
 *
 * Output arguments (not defined on input):
 *
 *       P: an int/SuiteSparse_long array of size n, containing the output
 *           permutation. If row i is the kth pivot row, then P [k] = i.  In
 *           MATLAB notation, the reordered matrix is A (P,P).
 *       ABIPInfo: a double array of size AMD_INFO, containing statistical
 *           information.  Ignored if ABIPInfo is ABIP_NULL.
 *
 * On input, the matrix A is stored in column-oriented form.  The row indices
 * of nonzero entries in column j are stored in Ai [Ap [j] ... Ap [j+1]-1].
 *
 * If the row indices appear in ascending order in each column, and there
 * are no duplicate entries, then amd_order is slightly more efficient in
 * terms of time and memory usage.  If this condition does not hold, a copy
 * of the matrix is created (where these conditions do hold), and the copy is
 * ordered.  This feature is new to v2.0 (v1.2 and earlier required this
 * condition to hold for the input matrix).
 *
 * Row indices must be in the range 0 to
 * n-1.  Ap [0] must be zero, and thus nz = Ap [n] is the number of nonzeros
 * in A.  The array Ap is of size n+1, and the array Ai is of size nz = Ap [n].
 * The matrix does not need to be symmetric, and the diagonal does not need to
 * be present (if diagonal entries are present, they are ignored except for
 * the output statistic Info [AMD_NZDIAG]).  The arrays Ai and Ap are not
 * modified.  This form of the Ap and Ai arrays to represent the nonzero
 * pattern of the matrix A is the same as that used internally by MATLAB.
 * If you wish to use a more flexible input structure, please see the
 * umfpack_*_triplet_to_col routines in the UMFPACK package, at
 * http://www.suitesparse.com.
 *
 * Restrictions:  n >= 0.  Ap [0] = 0.  Ap [j] <= Ap [j+1] for all j in the
 *       range 0 to n-1.  nz = Ap [n] >= 0.  Ai [0..nz-1] must be in the range 0
 *       to n-1.  Finally, Ai, Ap, and P must not be NULL.  If any of these
 *       restrictions are not met, AMD returns AMD_INVALID.
 *
 * AMD returns:
 *
 *       AMD_OK if the matrix is valid and sufficient memory can be allocated to
 *           perform the ordering.
 *
 *       AMD_OUT_OF_MEMORY if not enough memory can be allocated.
 *
```

```
 *       AMD_INVALID if the input arguments n, Ap, Ai are invalid, or if P is
 *           NULL.
 *
 *       AMD_OK_BUT_JUMBLED if the matrix had unsorted columns, and/or duplicate
 *           entries, but was otherwise valid.
 *
 * The AMD routine first forms the pattern of the matrix A+A', and then
 * computes a fill-reducing ordering, P.  If P [k] = i, then row/column i of
 * the original is the kth pivotal row.  In MATLAB notation, the permuted
 * matrix is A (P,P), except that 0-based indexing is used instead of the
 * 1-based indexing in MATLAB.
 *
 * The Control array is used to set various parameters for AMD.  If a NULL
 * pointer is passed, default values are used.  The Control array is not
 * modified.
 *
 *       Control [AMD_DENSE]:  controls the threshold for "dense" rows/columns.
 *           A dense row/column in A+A' can cause AMD to spend a lot of time in
 *           ordering the matrix.  If Control [AMD_DENSE] >= 0, rows/columns
 *           with more than Control [AMD_DENSE] * sqrt (n) entries are ignored
 *           during the ordering, and placed last in the output order.  The
 *           default value of Control [AMD_DENSE] is 10.  If negative, no
 *           rows/columns are treated as "dense".  Rows/columns with 16 or
 *           fewer off-diagonal entries are never considered "dense".
 *
 *       Control [AMD_AGGRESSIVE]: controls whether or not to use aggressive
 *           absorption, in which a prior element is absorbed into the current
 *           element if is a subset of the current element, even if it is not
 *           adjacent to the current pivot element (refer to Amestoy, Davis,
 *           & Duff, 1996, for more details).  The default value is nonzero,
 *           which means to perform aggressive absorption.  This nearly always
 *           leads to a better ordering (because the approximate degrees are
 *           more accurate) and a lower execution time.  There are cases where
 *           it can lead to a slightly worse ordering, however.  To turn it off,
 *           set Control [AMD_AGGRESSIVE] to 0.
 *
 *       Control [2..4] are not used in the current version, but may be used in
 *           future versions.
 *
 * The ABIPInfo array provides statistics about the ordering on output.  If it is
 * not present, the statistics are not returned.  This is not an error
 * condition.
 *
 *       ABIPInfo [AMD_STATUS]:  the return value of AMD, either AMD_OK,
 *           AMD_OK_BUT_JUMBLED, AMD_OUT_OF_MEMORY, or AMD_INVALID.
 *
 *       ABIPInfo [AMD_N]: n, the size of the input matrix
 *
 *       ABIPInfo [AMD_NZ]: the number of nonzeros in A, nz = Ap [n]
 *
 *       ABIPInfo [AMD_SYMMETRY]:  the symmetry of the matrix A.  It is the number
 *           of "matched" off-diagonal entries divided by the total number of
 *           off-diagonal entries.  An entry A(i,j) is matched if A(j,i) is also
 *           an entry, for any pair (i,j) for which i != j.  In MATLAB notation,
 *                S = spones (A) ;
 *                B = tril (S, -1) + triu (S, 1) ;
 *                symmetry = nnz (B & B') / nnz (B) ;
 *
 *       ABIPInfo [AMD_NZDIAG]: the number of entries on the diagonal of A.
 *
 *       ABIPInfo [AMD_NZ_A_PLUS_AT]:  the number of nonzeros in A+A', excluding the
 *           diagonal.  If A is perfectly symmetric (Info [AMD_SYMMETRY] = 1)
 *           with a fully nonzero diagonal, then Info [AMD_NZ_A_PLUS_AT] = nz-n
 *           (the smallest possible value).  If A is perfectly unsymmetric
 *           (Info [AMD_SYMMETRY] = 0, for an upper triangular matrix, for
 *           example) with no diagonal, then Info [AMD_NZ_A_PLUS_AT] = 2*nz
 *           (the largest possible value).
 *
 *       ABIPInfo [AMD_NDENSE]: the number of "dense" rows/columns of A+A' that were
 *           removed from A prior to ordering.  These are placed last in the
 *           output order P.
 *
 *       ABIPInfo [AMD_MEMORY]: the amount of memory used by AMD, in bytes.  In the
 *           current version, this is 1.2 * Info  [AMD_NZ_A_PLUS_AT] + 9*n
 *           times the size of an integer.  This is at most 2.4nz + 9n.  This
 *           excludes the size of the input arguments Ai, Ap, and P, which have
 *           a total size of nz + 2*n + 1 integers.
 *
 *       ABIPInfo [AMD_NCMPA]: the number of garbage collections performed.
 *
 *       ABIPInfo [AMD_LNZ]: the number of nonzeros in L (excluding the diagonal).
 *           This is a slight upper bound because mass elimination is combined
 *           with the approximate degree update.  It is a rough upper bound if
 *           there are many "dense" rows/columns.  The rest of the statistics,
 *           below, are also slight or rough upper bounds, for the same reasons.
 *           The post-ordering of the assembly tree might also not exactly
 *           correspond to a true elimination tree postordering.
```

```
 *
 *       ABIPInfo [AMD_NDIV]: the number of divide operations for a subsequent LDL'
 *           or LU factorization of the permuted matrix A (P,P).
 *
 *       ABIPInfo [AMD_NMULTSUBS_LDL]:  the number of multiply-subtract pairs for a
 *           subsequent LDL' factorization of A (P,P).
 *
 *       ABIPInfo [AMD_NMULTSUBS_LU]:  the number of multiply-subtract pairs for a
 *           subsequent LU factorization of A (P,P), assuming that no numerical
 *           pivoting is required.
 *
 *       ABIPInfo [AMD_DMAX]:  the maximum number of nonzeros in any column of L,
 *           including the diagonal.
 *
 *       ABIPInfo [14..19] are not used in the current version, but may be used in
 *           future versions.
 */

/* ------------------------------------------------------------------------- */
/* direct interface to AMD */
/* ------------------------------------------------------------------------- */

/* amd_2 is the primary AMD ordering routine.  It is not meant to be
 * user-callable because of its restrictive inputs and because it destroys
 * the user's input matrix.  It does not check its inputs for errors, either.
 * However, if you can work with these restrictions it can be faster than
 * amd_order and use less memory (assuming that you can create your own copy
 * of the matrix for AMD to destroy).  Refer to AMD/Source/amd_2.c for a
 * description of each parameter. */

void amd_2
(
            int n,
            int Pe [ ],
            int Iw [ ],
            int Len [ ],
            int iwlen,
            int pfree,
            int Nv [ ],
            int Next [ ],
            int Last [ ],
            int Head [ ],
            int Elen [ ],
            int Degree [ ],
            int W [ ],
            abip_float Control [ ],
            abip_float ABIPInfo [ ]
) ;

void amd_l2
(
            SuiteSparse_long n,
            SuiteSparse_long Pe [ ],
            SuiteSparse_long Iw [ ],
            SuiteSparse_long Len [ ],
            SuiteSparse_long iwlen,
            SuiteSparse_long pfree,
            SuiteSparse_long Nv [ ],
            SuiteSparse_long Next [ ],
            SuiteSparse_long Last [ ],
            SuiteSparse_long Head [ ],
            SuiteSparse_long Elen [ ],
            SuiteSparse_long Degree [ ],
            SuiteSparse_long W [ ],
            abip_float Control [ ],
            abip_float ABIPInfo [ ]
) ;

/* ------------------------------------------------------------------------- */
/* amd_valid */
/* ------------------------------------------------------------------------- */

/* Returns AMD_OK or AMD_OK_BUT_JUMBLED if the matrix is valid as input to
 * amd_order; the latter is returned if the matrix has unsorted and/or
 * duplicate row indices in one or more columns.  Returns AMD_INVALID if the
 * matrix cannot be passed to amd_order.  For amd_order, the matrix must also
 * be square.  The first two arguments are the number of rows and the number
 * of columns of the matrix.  For its use in AMD, these must both equal n.
 *
 * NOTE: this routine returned TRUE/FALSE in v1.2 and earlier.
 */

int amd_valid
(
            int n_row,                  /* # of rows */
            int n_col,                  /* # of columns */
            const int Ap [ ],          /* column pointers, of size n_col+1 */
```

```
                const int Ai [ ]            /* row indices, of size Ap [n_col] */
) ;

SuiteSparse_long amd_l_valid
(
                SuiteSparse_long n_row,
                SuiteSparse_long n_col,
                const SuiteSparse_long Ap [ ],
                const SuiteSparse_long Ai [ ]
) ;

/* ------------------------------------------------------------------------- */
/* AMD memory manager and printf routines */
/* ------------------------------------------------------------------------- */

#ifndef EXTERN
#define EXTERN extern
#endif

EXTERN void *(*amd_malloc) (size_t) ;                     /* pointer to malloc */
EXTERN void (*amd_free) (void *) ;                           /* pointer to free */
EXTERN void *(*amd_realloc) (void *, size_t) ;           /* pointer to realloc */
EXTERN void *(*amd_calloc) (size_t, size_t) ;             /* pointer to calloc */
EXTERN int (*amd_printf) (const char *, ...) ;              /* pointer to printf */

/* ------------------------------------------------------------------------- */
/* AMD Control and Info arrays */
/* ------------------------------------------------------------------------- */

/* amd_defaults:  sets the default control settings */
void amd_defaults   (abip_float Control [ ]) ;
void amd_l_defaults (abip_float Control [ ]) ;

/* amd_control: prints the control settings */
void amd_control    (abip_float Control [ ]) ;
void amd_l_control  (abip_float Control [ ]) ;

/* amd_info: prints the statistics */
void amd_info       (abip_float ABIPInfo [ ]) ;
void amd_l_info     (abip_float ABIPInfo [ ]) ;

#define AMD_CONTROL 5           /* size of Control array */
#define AMD_INFO 20             /* size of Info array */

/* contents of Control */
#define AMD_DENSE 0             /* "dense" if degree > Control [0] * sqrt (n) */
#define AMD_AGGRESSIVE 1    /* do aggressive absorption if Control [1] != 0 */

/* default Control settings */
#define AMD_DEFAULT_DENSE 10.0          /* default "dense" degree 10*sqrt(n) */
#define AMD_DEFAULT_AGGRESSIVE 1    /* do aggressive absorption by default */

/* contents of Info */
#define AMD_STATUS 0            /* return value of amd_order and amd_l_order */
#define AMD_N 1                 /* A is n-by-n */
#define AMD_NZ 2      /* number of nonzeros in A */
#define AMD_SYMMETRY 3          /* symmetry of pattern (1 is sym., 0 is unsym.) */
#define AMD_NZDIAG 4            /* # of entries on diagonal */
#define AMD_NZ_A_PLUS_AT 5  /* nz in A+A' */
#define AMD_NDENSE 6            /* number of "dense" rows/columns in A */
#define AMD_MEMORY 7            /* amount of memory used by AMD */
#define AMD_NCMPA 8             /* number of garbage collections in AMD */
#define AMD_LNZ 9     /* approx. nz in L, excluding the diagonal */
#define AMD_NDIV 10             /* number of fl. point divides for LU and LDL' */
#define AMD_NMULTSUBS_LDL 11 /* number of fl. point (*,-) pairs for LDL' */
#define AMD_NMULTSUBS_LU 12  /* number of fl. point (*,-) pairs for LU */
#define AMD_DMAX 13              /* max nz. in any column of L, incl. diagonal */

/* ------------------------------------------------------------------------- */
/* return values of AMD */
/* ------------------------------------------------------------------------- */

#define AMD_OK 0            /* success */
#define AMD_OUT_OF_MEMORY -1        /* malloc failed, or problem too large */
#define AMD_INVALID -2              /* input arguments are not valid */
#define AMD_OK_BUT_JUMBLED 1        /* input matrix is OK for amd_order, but
    * columns were not sorted, and/or duplicate entries were present.  AMD had
    * to do extra work before ordering the matrix.  This is a warning, not an
    * error.  */

/* ========================================================================== */
/* === AMD version ========================================================== */
/* ========================================================================== */

/* AMD Version 1.2 and later include the following definitions.
 * As an example, to test if the version you are using is 1.2 or later:
 *
```

```
 * #ifdef AMD_VERSION
 *       if (AMD_VERSION >= AMD_VERSION_CODE (1,2)) ...
 * #endif
 *
 * This also works during compile-time:
 *
 *       #if defined(AMD_VERSION) && (AMD_VERSION >= AMD_VERSION_CODE (1,2))
 *           printf ("This is version 1.2 or later\n") ;
 *       #else
 *           printf ("This is an early version\n") ;
 *       #endif
 *
 * Versions 1.1 and earlier of AMD do not include a #define'd version number.
 */

#define AMD_DATE "May 4, 2016"
#define AMD_VERSION_CODE(main,sub) ((main) * 1000 + (sub))
#define AMD_MAIN_VERSION 2
#define AMD_SUB_VERSION 4
#define AMD_SUBSUB_VERSION 6
#define AMD_VERSION AMD_VERSION_CODE(AMD_MAIN_VERSION,AMD_SUB_VERSION)

#ifdef __cplusplus
}
#endif

#endif
```

5.7 external/amd/amd_1.c File Reference

```
#include "amd_internal.h"
```

Functions

- GLOBAL void AMD_1 (Int n, const Int Ap[ ], const Int Ai[ ], Int P[ ], Int Pinv[ ], Int Len[ ], Int slen, Int S[ ], abip_float Control[ ], abip_float ABIPInfo[ ])

5.7.1 Function Documentation

5.7.1.1 AMD_1()

```
GLOBAL void AMD_1 (
            Int n,
            const Int Ap[ ],
            const Int Ai[ ],
            Int P[ ],
            Int Pinv[ ],
            Int Len[ ],
            Int slen,
            Int S[ ],
            abip_float Control[ ],
            abip_float ABIPInfo[ ] )
```

Definition at line 29 of file amd_1.c.

5.8 amd_1.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_1 =============================================================== */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                     */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* AMD_1: Construct A+A' for a sparse matrix A and perform the AMD ordering.
 *
 * The n-by-n sparse matrix A can be unsymmetric.  It is stored in MATLAB-style
 * compressed-column form, with sorted row indices in each column, and no
 * duplicate entries.  Diagonal entries may be present, but they are ignored.
 * Row indices of column j of A are stored in Ai [Ap [j] ... Ap [j+1]-1].
 * Ap [0] must be zero, and nz = Ap [n] is the number of entries in A.  The
 * size of the matrix, n, must be greater than or equal to zero.
 *
 * This routine must be preceded by a call to AMD_aat, which computes the
 * number of entries in each row/column in A+A', excluding the diagonal.
 * Len [j], on input, is the number of entries in row/column j of A+A'.  This
 * routine constructs the matrix A+A' and then calls AMD_2.  No error checking
 * is performed (this was done in AMD_valid).
 */

#include "amd_internal.h"

GLOBAL void AMD_1
(
        Int n,           /* n > 0 */
        const Int Ap [ ],    /* input of size n+1, not modified */
        const Int Ai [ ],        /* input of size nz = Ap [n], not modified */
        Int P [ ],       /* size n output permutation */
        Int Pinv [ ],        /* size n output inverse permutation */
        Int Len [ ],     /* size n input, undefined on output */
        Int slen,        /* slen >= sum (Len [0..n-1]) + 7n,
                 * ideally slen = 1.2 * sum (Len) + 8n */
        Int S [ ],       /* size slen workspace */
        abip_float Control [ ],    /* input array of size AMD_CONTROL */
        abip_float ABIPInfo [ ]    /* output array of size AMD_INFO */
)
{
        Int i;
        Int j;
        Int k;
        Int p;
        Int pfree;
        Int iwlen;
        Int pj;
        Int p1;
        Int p2;
        Int pj2;

        Int *Iw;
        Int *Pe;
        Int *Nv;
        Int *Head;
    Int *Elen;
    Int *Degree;
    Int *s;
    Int *W;
    Int *Sp;
    Int *Tp;

        /* --------------------------------------------------------------------- */
        /* construct the matrix for AMD_2 */
        /* --------------------------------------------------------------------- */

        ASSERT (n > 0) ;

        iwlen = slen - 6*n ;
        s = S ;
        Pe = s ;
        s += n ;
        Nv = s ;
        s += n ;
        Head = s ;
        s += n ;
        Elen = s ;
        s += n ;
        Degree = s ;
```

```
        s += n ;
        W = s ;
        s += n ;
        Iw = s ;
        s += iwlen ;

        ASSERT (AMD_valid (n, n, Ap, Ai) == AMD_OK) ;

        /* construct the pointers for A+A' */
        Sp = Nv ;           /* use Nv and W as workspace for Sp and Tp [ */
        Tp = W ;
        pfree = 0 ;

        for (j = 0 ; j < n ; j++)
        {
        Pe [j] = pfree ;
        Sp [j] = pfree ;
        pfree += Len [j] ;
        }

        /* Note that this restriction on iwlen is slightly more restrictive than
        * what is strictly required in AMD_2.  AMD_2 can operate with no elbow
        * room at all, but it will be very slow.  For better performance, at
        * least size-n elbow room is enforced. */
        ASSERT (iwlen >= pfree + n) ;

    #ifndef NDEBUG
        for (p = 0 ; p < iwlen ; p++) Iw [p] = EMPTY ;
    #endif

        for (k = 0 ; k < n ; k++)
        {
        AMD_DEBUG1 (("Construct row/column k= "ID" of A+A'\n", k))  ;
        p1 = Ap [k] ;
        p2 = Ap [k+1] ;

        /* construct A+A' */
        for (p = p1 ; p < p2 ; )
        {
                /* scan the upper triangular part of A */
                j = Ai [p] ;
                ASSERT (j >= 0 && j < n) ;

                if (j < k)
                {
                /* entry A (j,k) in the strictly upper triangular part */
                ASSERT (Sp [j] < (j == n-1 ? pfree : Pe [j+1])) ;
                ASSERT (Sp [k] < (k == n-1 ? pfree : Pe [k+1])) ;
                Iw [Sp [j]++] = k ;
                Iw [Sp [k]++] = j ;
                p++ ;
                }
                else if (j == k)
                {
                /* skip the diagonal */
                p++ ;
                break ;
                }
                else /* j > k */
                {
                /* first entry below the diagonal */
                break ;
                }

                /* scan lower triangular part of A, in column j until reaching
                * row k.  Start where last scan left off. */
                ASSERT (Ap [j] <= Tp [j] && Tp [j] <= Ap [j+1]) ;
                pj2 = Ap [j+1] ;

                for (pj = Tp [j] ; pj < pj2 ; )
                {
                i = Ai [pj] ;
                ASSERT (i >= 0 && i < n) ;

                if (i < k)
                {
                        /* A (i,j) is only in the lower part, not in upper */
                        ASSERT (Sp [i] < (i == n-1 ? pfree : Pe [i+1])) ;
                        ASSERT (Sp [j] < (j == n-1 ? pfree : Pe [j+1])) ;
                        Iw [Sp [i]++] = j ;
                        Iw [Sp [j]++] = i ;
                        pj++ ;
                }
                else if (i == k)
                {
                        /* entry A (k,j) in lower part and A (j,k) in upper */
                        pj++ ;
```

```
                        break ;
                }
                else /* i > k */
                {
                        /* consider this entry later, when k advances to i */
                        break ;
                }
                }
                Tp [j] = pj ;
        }
        Tp [k] = p ;
        }

        /* clean up, for remaining mismatched entries */
        for (j = 0 ; j < n ; j++)
        {
        for (pj = Tp [j] ; pj < Ap [j+1] ; pj++)
        {
                i = Ai [pj] ;
                ASSERT (i >= 0 && i < n) ;

                /* A (i,j) is only in the lower part, not in upper */
                ASSERT (Sp [i] < (i == n-1 ? pfree : Pe [i+1])) ;
                ASSERT (Sp [j] < (j == n-1 ? pfree : Pe [j+1])) ;
                Iw [Sp [i]++] = j ;
                Iw [Sp [j]++] = i ;
        }
        }

    #ifndef NDEBUG

        for (j = 0 ; j < n-1 ; j++) ASSERT (Sp [j] == Pe [j+1]) ;
        ASSERT (Sp [n-1] == pfree) ;

    #endif

        /* Tp and Sp no longer needed ] */

        /* --------------------------------------------------------------------- */
        /* order the matrix */
        /* --------------------------------------------------------------------- */

        AMD_2 (n, Pe, Iw, Len, iwlen, pfree, Nv, Pinv, P, Head, Elen, Degree, W, Control, ABIPInfo) ;
}
```

5.9 external/amd/amd_2.c File Reference

```
#include "amd_internal.h"
```

Functions

- GLOBAL void AMD_2 (Int n, Int Pe[ ], Int Iw[ ], Int Len[ ], Int iwlen, Int pfree, Int Nv[ ], Int Next[ ], Int Last[ ], Int Head[ ], Int Elen[ ], Int Degree[ ], Int W[ ], abip_float Control[ ], abip_float ABIPInfo[ ])

5.9.1 Function Documentation

5.9.1.1 AMD_2()

```
GLOBAL void AMD_2 (
        Int n,
        Int Pe[ ],
        Int Iw[ ],
```

```
Int Len[ ],
Int iwlen,
Int pfree,
Int Nv[ ],
Int Next[ ],
Int Last[ ],
Int Head[ ],
Int Elen[ ],
Int Degree[ ],
Int W[ ],
abip_float Control[ ],
abip_float ABIPInfo[ ] )
```

Definition at line 43 of file amd_2.c.

5.10 amd_2.c

Go to the documentation of this file.

```
00001 /* ========================================================================= */
00002 /* === AMD_2 =============================================================== */
00003 /* ========================================================================= */
00004
00005 /* ------------------------------------------------------------------------- */
00006 /* AMD, Copyright (c) Timothy A. Davis,                                    */
00007 /* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
00008 /* email: DrTimothyAldenDavis@gmail.com                                      */
00009 /* ------------------------------------------------------------------------- */
00010
00011 /* AMD_2:  performs the AMD ordering on a symmetric sparse matrix A, followed
00012  * by a postordering (via depth-first search) of the assembly tree using the
00013  * AMD_postorder routine.
00014  */
00015
00016 #include "amd_internal.h"
00017
00018 /* ========================================================================= */
00019 /* === clear_flag ========================================================== */
00020 /* ========================================================================= */
00021
00022 static Int clear_flag (Int wflg, Int wbig, Int W [ ], Int n)
00023 {
00024             Int x ;
00025             if (wflg < 2 || wflg >= wbig)
00026             {
00027                for (x = 0 ; x < n ; x++)
00028                {
00029                            if (W [x] != 0) W [x] = 1 ;
00030                }
00031                wflg = 2 ;
00032             }
00033
00034             /*  at this point, W [0..n-1] < wflg holds */
00035             return (wflg) ;
00036 }
00037
00038
00039 /* ========================================================================= */
00040 /* === AMD_2 =============================================================== */
00041 /* ========================================================================= */
00042
00043 GLOBAL void AMD_2
00044 (
00045             Int n,                   /* A is n-by-n, where n > 0 */
00046             Int Pe [ ],      /* Pe [0..n-1]: index in Iw of row i on input */
00047             Int Iw [ ],              /* workspace of size iwlen. Iw [0..pfree-1] holds the matrix on
      input */
00048             Int Len [ ],             /* Len [0..n-1]: length for row/column i on input */
00049             Int iwlen,               /* length of Iw. iwlen >= pfree + n */
00050             Int pfree,       /* Iw [pfree ... iwlen-1] is empty on input */
00051
00052             /* 7 size-n workspaces, not defined on input: */
00053             Int Nv [ ],      /* the size of each supernode on output */
00054             Int Next [ ],            /* the output inverse permutation */
00055             Int Last [ ],            /* the output permutation */
00056             Int Head [ ],
```

```
            Int Elen [ ],                 /* the size columns of L for each supernode */
            Int Degree [ ],
            Int W [ ],

            /* control parameters and output statistics */
            abip_float Control [ ],    /* array of size AMD_CONTROL */
            abip_float ABIPInfo [ ]    /* array of size AMD_INFO */
)
{

/*
 * Given a representation of the nonzero pattern of a symmetric matrix, A,
 * (excluding the diagonal) perform an approximate minimum (UMFPACK/MA38-style)
 * degree ordering to compute a pivot order such that the introduction of
 * nonzeros (fill-in) in the Cholesky factors A = LL' is kept low.  At each
 * step, the pivot selected is the one with the minimum UMFAPACK/MA38-style
 * upper-bound on the external degree.  This routine can optionally perform
 * aggresive absorption (as done by MC47B in the Harwell Subroutine
 * Library).
 *
 * The approximate degree algorithm implemented here is the symmetric analog of
 * the degree update algorithm in MA38 and UMFPACK (the Unsymmetric-pattern
 * MultiFrontal PACKage, both by Davis and Duff).  The routine is based on the
 * MA27 minimum degree ordering algorithm by Iain Duff and John Reid.
 *
 * This routine is a translation of the original AMDBAR and MC47B routines,
 * in Fortran, with the following modifications:
 *
 * (1) dense rows/columns are removed prior to ordering the matrix, and placed
 *  last in the output order.  The presence of a dense row/column can
 *  increase the ordering time by up to O(n^2), unless they are removed
 *  prior to ordering.
 *
 * (2) the minimum degree ordering is followed by a postordering (depth-first
 *  search) of the assembly tree.  Note that mass elimination (discussed
 *  below) combined with the approximate degree update can lead to the mass
 *  elimination of nodes with lower exact degree than the current pivot
 *  element.  No additional fill-in is caused in the representation of the
 *  Schur complement.  The mass-eliminated nodes merge with the current
 *  pivot element.  They are ordered prior to the current pivot element.
 *  Because they can have lower exact degree than the current element, the
 *  merger of two or more of these nodes in the current pivot element can
 *  lead to a single element that is not a "fundamental supernode".  The
 *  diagonal block can have zeros in it.  Thus, the assembly tree used here
 *  is not guaranteed to be the precise supernodal elemination tree (with
 *  "funadmental" supernodes), and the postordering performed by this
 *  routine is not guaranteed to be a precise postordering of the
 *  elimination tree.
 *
 * (3) input parameters are added, to control aggressive absorption and the
 *  detection of "dense" rows/columns of A.
 *
 * (4) additional statistical information is returned, such as the number of
 *  nonzeros in L, and the flop counts for subsequent LDL' and LU
 *  factorizations.  These are slight upper bounds, because of the mass
 *  elimination issue discussed above.
 *
 * (5) additional routines are added to interface this routine to MATLAB
 *  to provide a simple C-callable user-interface, to check inputs for
 *  errors, compute the symmetry of the pattern of A and the number of
 *  nonzeros in each row/column of A+A', to compute the pattern of A+A',
 *  to perform the assembly tree postordering, and to provide debugging
 *  ouput.  Many of these functions are also provided by the Fortran
 *  Harwell Subroutine Library routine MC47A.
 *
 * (6) both int and SuiteSparse_long versions are provided.  In the
 *      descriptions below and integer is and int or SuiteSparse_long depending
 *      on which version is being used.

 **********************************************************************
 ***** CAUTION:  ARGUMENTS ARE NOT CHECKED FOR ERRORS ON INPUT.  ******
 **********************************************************************
 ** If you want error checking, a more versatile input format, and a **
 ** simpler user interface, use amd_order or amd_l_order instead.    **
 ** This routine is not meant to be user-callable.                   **
 **********************************************************************

 * ----------------------------------------------------------------------------
 * References:
 * ----------------------------------------------------------------------------
 *
 *  [1] Timothy A. Davis and Iain Duff, "An unsymmetric-pattern multifrontal
 *  method for sparse LU factorization", SIAM J. Matrix Analysis and
 *  Applications, vol. 18, no. 1, pp. 140-158.  Discusses UMFPACK / MA38,
 *  which first introduced the approximate minimum degree used by this
 *  routine.
 *
```

```
 *  [2] Patrick Amestoy, Timothy A. Davis, and Iain S. Duff, "An approximate
 *  minimum degree ordering algorithm," SIAM J. Matrix Analysis and
 *  Applications, vol. 17, no. 4, pp. 886-905, 1996.  Discusses AMDBAR and
 *  MC47B, which are the Fortran versions of this routine.
 *
 *  [3] Alan George and Joseph Liu, "The evolution of the minimum degree
 *  ordering algorithm," SIAM Review, vol. 31, no. 1, pp. 1-19, 1989.
 *  We list below the features mentioned in that paper that this code
 *  includes:
 *
 *  mass elimination:
 *      Yes.  MA27 relied on supervariable detection for mass elimination.
 *
 *  indistinguishable nodes:
 *      Yes (we call these "supervariables").  This was also in the MA27
 *      code - although we modified the method of detecting them (the
 *      previous hash was the true degree, which we no longer keep track
 *      of).  A supervariable is a set of rows with identical nonzero
 *      pattern.  All variables in a supervariable are eliminated together.
 *      Each supervariable has as its numerical name that of one of its
 *      variables (its principal variable).
 *
 *  quotient graph representation:
 *      Yes.  We use the term "element" for the cliques formed during
 *      elimination.  This was also in the MA27 code.  The algorithm can
 *      operate in place, but it will work more efficiently if given some
 *      "elbow room."
 *
 *  element absorption:
 *      Yes.  This was also in the MA27 code.
 *
 *  external degree:
 *      Yes.  The MA27 code was based on the true degree.
 *
 *  incomplete degree update and multiple elimination:
 *      No.  This was not in MA27, either.  Our method of degree update
 *      within MC47B is element-based, not variable-based.  It is thus
 *      not well-suited for use with incomplete degree update or multiple
 *      elimination.
 *
 * Authors, and Copyright (C) 2004 by:
 * Timothy A. Davis, Patrick Amestoy, Iain S. Duff, John K. Reid.
 *
 * Acknowledgements: This work (and the UMFPACK package) was supported by the
 * National Science Foundation (ASC-9111263, DMS-9223088, and CCR-0203270).
 * The UMFPACK/MA38 approximate degree update algorithm, the unsymmetric analog
 * which forms the basis of AMD, was developed while Tim Davis was supported by
 * CERFACS (Toulouse, France) in a post-doctoral position.  This C version, and
 * the etree postorder, were written while Tim Davis was on sabbatical at
 * Stanford University and Lawrence Berkeley National Laboratory.

 * ----------------------------------------------------------------------------
 * INPUT ARGUMENTS (unaltered):
 * ----------------------------------------------------------------------------

 * n:  The matrix order.  Restriction:  n >= 1.
 *
 * iwlen:  The size of the Iw array.  On input, the matrix is stored in
 *  Iw [0..pfree-1].  However, Iw [0..iwlen-1] should be slightly larger
 *  than what is required to hold the matrix, at least iwlen >= pfree + n.
 *  Otherwise, excessive compressions will take place.  The recommended
 *  value of iwlen is 1.2 * pfree + n, which is the value used in the
 *  user-callable interface to this routine (amd_order.c).  The algorithm
 *  will not run at all if iwlen < pfree.  Restriction: iwlen >= pfree + n.
 *  Note that this is slightly more restrictive than the actual minimum
 *  (iwlen >= pfree), but AMD_2 will be very slow with no elbow room.
 *  Thus, this routine enforces a bare minimum elbow room of size n.
 *
 * pfree: On input the tail end of the array, Iw [pfree..iwlen-1], is empty,
 *  and the matrix is stored in Iw [0..pfree-1].  During execution,
 *  additional data is placed in Iw, and pfree is modified so that
 *  Iw [pfree..iwlen-1] is always the unused part of Iw.
 *
 * Control:  A abip_float array of size AMD_CONTROL containing input parameters
 *  that affect how the ordering is computed.  If ABIP_NULL, then default
 *  settings are used.
 *
 *  Control [AMD_DENSE] is used to determine whether or not a given input
 *  row is "dense".  A row is "dense" if the number of entries in the row
 *  exceeds Control [AMD_DENSE] times sqrt (n), except that rows with 16 or
 *  fewer entries are never considered "dense".  To turn off the detection
 *  of dense rows, set Control [AMD_DENSE] to a negative number, or to a
 *  number larger than sqrt (n).  The default value of Control [AMD_DENSE]
 *  is AMD_DEFAULT_DENSE, which is defined in amd.h as 10.
 *
 *  Control [AMD_AGGRESSIVE] is used to determine whether or not aggressive
 *  absorption is to be performed.  If nonzero, then aggressive absorption
```

```
 *  is performed (this is the default).

 * ----------------------------------------------------------------------------
 * INPUT/OUPUT ARGUMENTS:
 * ----------------------------------------------------------------------------
 *
 * Pe:  An integer array of size n.  On input, Pe [i] is the index in Iw of
 *  the start of row i.  Pe [i] is ignored if row i has no off-diagonal
 *  entries.  Thus Pe [i] must be in the range 0 to pfree-1 for non-empty
 *  rows.
 *
 *  During execution, it is used for both supervariables and elements:
 *
 *  Principal supervariable i:  index into Iw of the description of
 *      supervariable i.  A supervariable represents one or more rows of
 *      the matrix with identical nonzero pattern.  In this case,
 *      Pe [i] >= 0.
 *
 *  Non-principal supervariable i:  if i has been absorbed into another
 *      supervariable j, then Pe [i] = FLIP (j), where FLIP (j) is defined
 *      as (-(j)-2).  Row j has the same pattern as row i.  Note that j
 *      might later be absorbed into another supervariable j2, in which
 *      case Pe [i] is still FLIP (j), and Pe [j] = FLIP (j2) which is
 *      < EMPTY, where EMPTY is defined as (-1) in amd_internal.h.
 *
 *  Unabsorbed element e:  the index into Iw of the description of element
 *      e, if e has not yet been absorbed by a subsequent element.  Element
 *      e is created when the supervariable of the same name is selected as
 *      the pivot.  In this case, Pe [i] >= 0.
 *
 *  Absorbed element e:  if element e is absorbed into element e2, then
 *      Pe [e] = FLIP (e2).  This occurs when the pattern of e (which we
 *      refer to as Le) is found to be a subset of the pattern of e2 (that
 *      is, Le2).  In this case, Pe [i] < EMPTY.  If element e is "null"
 *      (it has no nonzeros outside its pivot block), then Pe [e] = EMPTY,
 *      and e is the root of an assembly subtree (or the whole tree if
 *      there is just one such root).
 *
 *  Dense variable i:  if i is "dense", then Pe [i] = EMPTY.
 *
 *  On output, Pe holds the assembly tree/forest, which implicitly
 *  represents a pivot order with identical fill-in as the actual order
 *  (via a depth-first search of the tree), as follows.  If Nv [i] > 0,
 *  then i represents a node in the assembly tree, and the parent of i is
 *  Pe [i], or EMPTY if i is a root.  If Nv [i] = 0, then (i, Pe [i])
 *  represents an edge in a subtree, the root of which is a node in the
 *  assembly tree.  Note that i refers to a row/column in the original
 *  matrix, not the permuted matrix.
 *
 * ABIPInfo:  A abip_float array of size AMD_INFO.  If present, (that is, not ABIP_NULL),
 *  then statistics about the ordering are returned in the ABIPInfo array.
 *  See amd.h for a description.

 * ----------------------------------------------------------------------------
 * INPUT/MODIFIED (undefined on output):
 * ----------------------------------------------------------------------------
 *
 * Len:  An integer array of size n.  On input, Len [i] holds the number of
 *  entries in row i of the matrix, excluding the diagonal.  The contents
 *  of Len are undefined on output.
 *
 * Iw:  An integer array of size iwlen.  On input, Iw [0..pfree-1] holds the
 *  description of each row i in the matrix.  The matrix must be symmetric,
 *  and both upper and lower triangular parts must be present.  The
 *  diagonal must not be present.  Row i is held as follows:
 *
 *      Len [i]:  the length of the row i data structure in the Iw array.
 *      Iw [Pe [i] ... Pe [i] + Len [i] - 1]:
 *      the list of column indices for nonzeros in row i (simple
 *      supervariables), excluding the diagonal.  All supervariables
 *      start with one row/column each (supervariable i is just row i).
 *      If Len [i] is zero on input, then Pe [i] is ignored on input.
 *
 *      Note that the rows need not be in any particular order, and there
 *      may be empty space between the rows.
 *
 *  During execution, the supervariable i experiences fill-in.  This is
 *  represented by placing in i a list of the elements that cause fill-in
 *  in supervariable i:
 *
 *      Len [i]:  the length of supervariable i in the Iw array.
 *      Iw [Pe [i] ... Pe [i] + Elen [i] - 1]:
 *      the list of elements that contain i.  This list is kept short
 *      by removing absorbed elements.
 *      Iw [Pe [i] + Elen [i] ... Pe [i] + Len [i] - 1]:
 *      the list of supervariables in i.  This list is kept short by
 *      removing nonprincipal variables, and any entry j that is also
```

```
 *      contained in at least one of the elements (j in Le) in the list
 *      for i (e in row i).
 *
 *  When supervariable i is selected as pivot, we create an element e of
 *  the same name (e=i):
 *
 *      Len [e]:  the length of element e in the Iw array.
 *      Iw [Pe [e] ... Pe [e] + Len [e] - 1]:
 *      the list of supervariables in element e.
 *
 *  An element represents the fill-in that occurs when supervariable i is
 *  selected as pivot (which represents the selection of row i and all
 *  non-principal variables whose principal variable is i).  We use the
 *  term Le to denote the set of all supervariables in element e.  Absorbed
 *  supervariables and elements are pruned from these lists when
 *  computationally convenient.
 *
 *  CAUTION:  THE INPUT MATRIX IS OVERWRITTEN DURING COMPUTATION.
 *  The contents of Iw are undefined on output.

 * ----------------------------------------------------------------------------
 * OUTPUT (need not be set on input):
 * ----------------------------------------------------------------------------
 *
 * Nv:  An integer array of size n.  During execution, ABS (Nv [i]) is equal to
 *  the number of rows that are represented by the principal supervariable
 *  i.  If i is a nonprincipal or dense variable, then Nv [i] = 0.
 *  Initially, Nv [i] = 1 for all i.  Nv [i] < 0 signifies that i is a
 *  principal variable in the pattern Lme of the current pivot element me.
 *  After element me is constructed, Nv [i] is set back to a positive
 *  value.
 *
 *  On output, Nv [i] holds the number of pivots represented by super
 *  row/column i of the original matrix, or Nv [i] = 0 for non-principal
 *  rows/columns.  Note that i refers to a row/column in the original
 *  matrix, not the permuted matrix.
 *
 * Elen:  An integer array of size n.  See the description of Iw above.  At the
 *  start of execution, Elen [i] is set to zero for all rows i.  During
 *  execution, Elen [i] is the number of elements in the list for
 *  supervariable i.  When e becomes an element, Elen [e] = FLIP (esize) is
 *  set, where esize is the size of the element (the number of pivots, plus
 *  the number of nonpivotal entries).  Thus Elen [e] < EMPTY.
 *  Elen (i) = EMPTY set when variable i becomes nonprincipal.
 *
 *  For variables, Elen (i) >= EMPTY holds until just before the
 *  postordering and permutation vectors are computed.  For elements,
 *  Elen [e] < EMPTY holds.
 *
 *  On output, Elen [i] is the degree of the row/column in the Cholesky
 *  factorization of the permuted matrix, corresponding to the original row
 *  i, if i is a super row/column.  It is equal to EMPTY if i is
 *  non-principal.  Note that i refers to a row/column in the original
 *  matrix, not the permuted matrix.
 *
 *  Note that the contents of Elen on output differ from the Fortran
 *  version (Elen holds the inverse permutation in the Fortran version,
 *  which is instead returned in the Next array in this C version,
 *  described below).
 *
 * Last: In a degree list, Last [i] is the supervariable preceding i, or EMPTY
 *  if i is the head of the list.  In a hash bucket, Last [i] is the hash
 *  key for i.
 *
 *  Last [Head [hash]] is also used as the head of a hash bucket if
 *  Head [hash] contains a degree list (see the description of Head,
 *  below).
 *
 *  On output, Last [0..n-1] holds the permutation.  That is, if
 *  i = Last [k], then row i is the kth pivot row (where k ranges from 0 to
 *  n-1).  Row Last [k] of A is the kth row in the permuted matrix, PAP'.
 *
 * Next: Next [i] is the supervariable following i in a link list, or EMPTY if
 *  i is the last in the list.  Used for two kinds of lists:  degree lists
 *  and hash buckets (a supervariable can be in only one kind of list at a
 *  time).
 *
 *  On output Next [0..n-1] holds the inverse permutation.  That is, if
 *  k = Next [i], then row i is the kth pivot row. Row i of A appears as
 *  the (Next[i])-th row in the permuted matrix, PAP'.
 *
 *  Note that the contents of Next on output differ from the Fortran
 *  version (Next is undefined on output in the Fortran version).

 * ----------------------------------------------------------------------------
 * LOCAL WORKSPACE (not input or output - used only during execution):
 * ----------------------------------------------------------------------------
```

```
 *
 * Degree:  An integer array of size n.  If i is a supervariable, then
 *  Degree [i] holds the current approximation of the external degree of
 *  row i (an upper bound).  The external degree is the number of nonzeros
 *  in row i, minus ABS (Nv [i]), the diagonal part.  The bound is equal to
 *  the exact external degree if Elen [i] is less than or equal to two.
 *
 *  We also use the term "external degree" for elements e to refer to
 *  |Le \ Lme|.  If e is an element, then Degree [e] is |Le|, which is the
 *  degree of the off-diagonal part of the element e (not including the
 *  diagonal part).
 *
 * Head:   An integer array of size n.  Head is used for degree lists.
 *  Head [deg] is the first supervariable in a degree list.  All
 *  supervariables i in a degree list Head [deg] have the same approximate
 *  degree, namely, deg = Degree [i].  If the list Head [deg] is empty then
 *  Head [deg] = EMPTY.
 *
 *  During supervariable detection Head [hash] also serves as a pointer to
 *  a hash bucket.  If Head [hash] >= 0, there is a degree list of degree
 *  hash.  The hash bucket head pointer is Last [Head [hash]].  If
 *  Head [hash] = EMPTY, then the degree list and hash bucket are both
 *  empty.  If Head [hash] < EMPTY, then the degree list is empty, and
 *  FLIP (Head [hash]) is the head of the hash bucket.  After supervariable
 *  detection is complete, all hash buckets are empty, and the
 *  (Last [Head [hash]] = EMPTY) condition is restored for the non-empty
 *  degree lists.
 *
 * W:  An integer array of size n.  The flag array W determines the status of
 *  elements and variables, and the external degree of elements.
 *
 *  for elements:
 *      if W [e] = 0, then the element e is absorbed.
 *      if W [e] >= wflg, then W [e] - wflg is the size of the set
 *      |Le \ Lme|, in terms of nonzeros (the sum of ABS (Nv [i]) for
 *      each principal variable i that is both in the pattern of
 *      element e and NOT in the pattern of the current pivot element,
 *      me).
 *      if wflg > W [e] > 0, then e is not absorbed and has not yet been
 *      seen in the scan of the element lists in the computation of
 *      |Le\Lme| in Scan 1 below.
 *
 *  for variables:
 *      during supervariable detection, if W [j] != wflg then j is
 *      not in the pattern of variable i.
 *
 *  The W array is initialized by setting W [i] = 1 for all i, and by
 *  setting wflg = 2.  It is reinitialized if wflg becomes too large (to
 *  ensure that wflg+n does not cause integer overflow).

 * ----------------------------------------------------------------------------
 * LOCAL INTEGERS:
 * ----------------------------------------------------------------------------
 */

          Int deg;
          Int degme;
          Int dext;
          Int lemax;

          Int e;
          Int elenme;
          Int eln;

          Int i;
          Int ilast;
          Int inext;

          Int j;
          Int jlast;
          Int jnext;

          Int k;
          Int knt1;
          Int knt2;
          Int knt3;

          Int lenj;
          Int ln;
          Int me;
          Int mindeg;
          Int nel;
          Int nleft;
          Int nvi;
          Int nvj;
          Int nvpiv;
          Int slenme;
```

```

            Int wbig;
            Int we;
            Int wflg;
            Int wnvi;

            Int ok;
            Int ndense;
            Int ncmpa;
            Int dense;
            Int aggressive;

            unsigned Int hash ;      /* unsigned, so that hash % n is well defined.*/

/*
 * deg:       the degree of a variable or element
 * degme:     size, |Lme|, of the current element, me (= Degree [me])
 * dext:      external degree, |Le \ Lme|, of some element e
 * lemax:     largest |Le| seen so far (called dmax in Fortran version)
 * e:         an element
 * elenme:    the length, Elen [me], of element list of pivotal variable
 * eln:       the length, Elen [...], of an element list
 * hash:      the computed value of the hash function
 * i:         a supervariable
 * ilast:     the entry in a link list preceding i
 * inext:     the entry in a link list following i
 * j:         a supervariable
 * jlast:     the entry in a link list preceding j
 * jnext:     the entry in a link list, or path, following j
 * k:         the pivot order of an element or variable
 * knt1:      loop counter used during element construction
 * knt2:      loop counter used during element construction
 * knt3:      loop counter used during compression
 * lenj:      Len [j]
 * ln:        length of a supervariable list
 * me:        current supervariable being eliminated, and the current
 *            element created by eliminating that supervariable
 * mindeg:    current minimum degree
 * nel:       number of pivots selected so far
 * nleft:     n - nel, the number of nonpivotal rows/columns remaining
 * nvi:       the number of variables in a supervariable i (= Nv [i])
 * nvj:       the number of variables in a supervariable j (= Nv [j])
 * nvpiv:     number of pivots in current element
 * slenme:    number of variables in variable list of pivotal variable
 * wbig:      = (INT_MAX - n) for the int version, (SuiteSparse_long_max - n)
 *                    for the SuiteSparse_long version.  wflg is not allowed to
 *                    be >= wbig.
 * we:        W [e]
 * wflg:      used for flagging the W array.  See description of Iw.
 * wnvi:      wflg - Nv [i]
 * x:         either a supervariable or an element
 *
 * ok:        true if supervariable j can be absorbed into i
 * ndense:    number of "dense" rows/columns
 * dense:     rows/columns with initial degree > dense are considered "dense"
 * aggressive:  true if aggressive absorption is being performed
 * ncmpa:     number of garbage collections

 * ----------------------------------------------------------------------------
 * LOCAL DOUBLES, used for statistical output only (except for alpha):
 * ----------------------------------------------------------------------------
 */

            abip_float f;
            abip_float r;
            abip_float ndiv;
            abip_float s;
            abip_float nms_lu;
            abip_float nms_ldl;
            abip_float dmax;
            abip_float alpha;
            abip_float lnz;
            abip_float lnzme;

/*
 * f:         nvpiv
 * r:         degme + nvpiv
 * ndiv:      number of divisions for LU or LDL' factorizations
 * s:         number of multiply-subtract pairs for LU factorization, for the
 *            current element me
 * nms_lu     number of multiply-subtract pairs for LU factorization
 * nms_ldl    number of multiply-subtract pairs for LDL' factorization
 * dmax:      the largest number of entries in any column of L, including the
 *            diagonal
 * alpha:     "dense" degree ratio
 * lnz:       the number of nonzeros in L (excluding the diagonal)
 * lnzme:     the number of nonzeros in L (excl. the diagonal) for the
```

```
 *          current element me

 * ----------------------------------------------------------------------------
 * LOCAL "POINTERS" (indices into the Iw array)
 * ----------------------------------------------------------------------------
*/

            Int p;
            Int p1;
            Int p2;
            Int p3;
            Int p4;
            Int pdst;
            Int pend;
            Int pj;
            Int pme;
            Int pme1;
            Int pme2;
            Int pn;
            Int psrc;

/*
 * Any parameter (Pe [...] or pfree) or local variable starting with "p" (for
 * Pointer) is an index into Iw, and all indices into Iw use variables starting
 * with "p."  The only exception to this rule is the iwlen input argument.
 *
 * p:           pointer into lots of things
 * p1:          Pe [i] for some variable i (start of element list)
 * p2:          Pe [i] + Elen [i] -  1 for some variable i
 * p3:          index of first supervariable in clean list
 * p4:
 * pdst:        destination pointer, for compression
 * pend:        end of memory to compress
 * pj:          pointer into an element or variable
 * pme:         pointer into the current element (pme1...pme2)
 * pme1:        the current element, me, is stored in Iw [pme1...pme2]
 * pme2:        the end of the current element
 * pn:          pointer into a "clean" variable, also used to compress
 * psrc:        source pointer, for compression
*/

/* ========================================================================= */
/*  INITIALIZATIONS */
/* ========================================================================= */

            /* Note that this restriction on iwlen is slightly more restrictive than
            * what is actually required in AMD_2.  AMD_2 can operate with no elbow
            * room at all, but it will be slow.  For better performance, at least
            * size-n elbow room is enforced. */
            ASSERT (iwlen >= pfree + n) ;
            ASSERT (n > 0) ;

            /* initialize output statistics */
            lnz = 0 ;
            ndiv = 0 ;
            nms_lu = 0 ;
            nms_ldl = 0 ;
            dmax = 1 ;
            me = EMPTY ;

            mindeg = 0 ;
            ncmpa = 0 ;
            nel = 0 ;
            lemax = 0 ;

            /* get control parameters */
            if (Control != (abip_float *) ABIP_NULL)
            {
               alpha = Control [AMD_DENSE] ;
               aggressive = (Control [AMD_AGGRESSIVE] != 0) ;
            }
            else
            {
               alpha = AMD_DEFAULT_DENSE ;
               aggressive = AMD_DEFAULT_AGGRESSIVE ;
            }

            /* Note: if alpha is NaN, this is undefined: */
            if (alpha < 0)
            {
               /* only remove completely dense rows/columns */
               dense = n-2 ;
            }
            else
            {
               dense = alpha * sqrt ((abip_float) n) ;
            }
```

```

    dense = MAX (16, dense) ;
    dense = MIN (n,  dense) ;
    AMD_DEBUG1 (("\n\nAMD (debug), alpha %g, aggr. "ID"\n", alpha, aggressive)) ;

    for (i = 0 ; i < n ; i++)
    {
       Last [i] = EMPTY ;
       Head [i] = EMPTY ;
       Next [i] = EMPTY ;

                /* if separate Hhead array is used for hash buckets: Hhead [i] = EMPTY ; */
       Nv [i] = 1 ;
       W [i] = 1 ;
       Elen [i] = 0 ;
       Degree [i] = Len [i] ;
    }

    #ifndef NDEBUG

    AMD_DEBUG1 (("\n======Nel "ID" initial\n", nel)) ;
    AMD_dump (n, Pe, Iw, Len, iwlen, pfree, Nv, Next, Last, Head, Elen, Degree, W, -1) ;

    #endif

    /* initialize wflg */
    wbig = Int_MAX - n ;
    wflg = clear_flag (0, wbig, W, n) ;

    /* --------------------------------------------------------------------- */
    /* initialize degree lists and eliminate dense and empty rows */
    /* --------------------------------------------------------------------- */

    ndense = 0 ;

    for (i = 0 ; i < n ; i++)
    {
       deg = Degree [i] ;
       ASSERT (deg >= 0 && deg < n) ;

                if (deg == 0)
       {
                            /* -------------------------------------------------------------
                   * we have a variable that can be eliminated at once because
                   * there is no off-diagonal non-zero in its row.  Note that
                   * Nv [i] = 1 for an empty variable i.  It is treated just
                   * the same as an eliminated element i.
                   * ------------------------------------------------------------- */

                   Elen [i] = FLIP (1) ;
                   nel++ ;
                   Pe [i] = EMPTY ;
                   W [i] = 0 ;
                }
       else if (deg > dense)
       {
                            /* -------------------------------------------------------------
                   * Dense variables are not treated as elements, but as unordered,
                   * non-principal variables that have no parent.  They do not take
                   * part in the postorder, since Nv [i] = 0.  Note that the Fortran
                   * version does not have this option.
                   * ------------------------------------------------------------- */

                   AMD_DEBUG1 (("Dense node "ID" degree "ID"\n", i, deg)) ;
                   ndense++ ;
                   Nv [i] = 0 ;      /* do not postorder this node */
                   Elen [i] = EMPTY ;
                   nel++ ;
                   Pe [i] = EMPTY ;
       }
       else
       {
                            /* -------------------------------------------------------------
                   * place i in the degree list corresponding to its degree
                   * ------------------------------------------------------------- */

                   inext = Head [deg] ;
                   ASSERT (inext >= EMPTY && inext < n) ;
                   if (inext != EMPTY) Last [inext] = i ;
                   Next [i] = inext ;
                   Head [deg] = i ;
                }
    }

    /* ===================================================================== */
    /* WHILE (selecting pivots) DO */
    /* ===================================================================== */
```

```

        while (nel < n)
        {

                    #ifndef NDEBUG

                    AMD_DEBUG1 (("\n======Nel "ID"\n", nel)) ;
           if (AMD_debug >= 2)
           {
                       AMD_dump (n, Pe, Iw, Len, iwlen, pfree, Nv, Next, Last, Head, Elen, Degree, W, nel) ;
           }

                    #endif

                    /* ========================================================================= */
                    /* GET PIVOT OF MINIMUM DEGREE */
                    /* ========================================================================= */

           /* ----------------------------------------------------------------- */
           /* find next supervariable for elimination */
           /* ----------------------------------------------------------------- */

           ASSERT (mindeg >= 0 && mindeg < n) ;
           for (deg = mindeg ; deg < n ; deg++)
           {
                       me = Head [deg] ;
                       if (me != EMPTY) break ;
           }
           mindeg = deg ;
           ASSERT (me >= 0 && me < n) ;
           AMD_DEBUG1 (("=================me: "ID"\n", me)) ;

           /* ----------------------------------------------------------------- */
           /* remove chosen variable from link list */
           /* ----------------------------------------------------------------- */

           inext = Next [me] ;
           ASSERT (inext >= EMPTY && inext < n) ;
           if (inext != EMPTY) Last [inext] = EMPTY ;
           Head [deg] = inext ;

           /* ----------------------------------------------------------------- */
           /* me represents the elimination of pivots nel to nel+Nv[me]-1. */
           /* place me itself as the first in this set. */
           /* ----------------------------------------------------------------- */

           elenme = Elen [me] ;
           nvpiv = Nv [me] ;
           ASSERT (nvpiv > 0) ;
           nel += nvpiv ;

                    /* ========================================================================= */
                    /* CONSTRUCT NEW ELEMENT */
                    /* ========================================================================= */

           /* -----------------------------------------------------------------
           * At this point, me is the pivotal supervariable.  It will be
           * converted into the current element.  Scan list of the pivotal
           * supervariable, me, setting tree pointers and constructing new list
           * of supervariables for the new element, me.  p is a pointer to the
           * current position in the old list.
           * ----------------------------------------------------------------- */

           /* flag the variable "me" as being in Lme by negating Nv [me] */
           Nv [me] = -nvpiv ;
           degme = 0 ;
           ASSERT (Pe [me] >= 0 && Pe [me] < iwlen) ;

           if (elenme == 0)
           {
                                /* ------------------------------------------------------------ */
                       /* construct the new element in place */
                       /* ------------------------------------------------------------ */

                       pme1 = Pe [me] ;
                       pme2 = pme1 - 1 ;

                       for (p = pme1 ; p <= pme1 + Len [me] - 1 ; p++)
                       {
                          i = Iw [p] ;
                          ASSERT (i >= 0 && i < n && Nv [i] >= 0) ;
```

```
                                nvi = Nv [i] ;

                                                        if (nvi > 0)
                                  {
                                                                        /* ------------------------------------------------------ */
                                                /* i is a principal variable not yet placed in Lme. */
                                                /* store i in new list */
                                                /* ------------------------------------------------------ */

                                                /* flag i as being in Lme by negating Nv [i] */
                                                degme += nvi ;
                                                Nv [i] = -nvi ;
                                                Iw [++pme2] = i ;

                                                /* ------------------------------------------------------ */
                                                /* remove variable i from degree list. */
                                                /* ------------------------------------------------------ */
                                                                ilast = Last [i] ;
                                                                inext = Next [i] ;
                                                ASSERT (ilast >= EMPTY && ilast < n) ;
                                                ASSERT (inext >= EMPTY && inext < n) ;
                                                if (inext != EMPTY) Last [inext] = ilast ;
                                                if (ilast != EMPTY)
                                                {
                                                   Next [ilast] = inext ;
                                                }
                                                else
                                                {
                                                   /* i is at the head of the degree list */
                                                   ASSERT (Degree [i] >= 0 && Degree [i] < n) ;
                                                   Head [Degree [i]] = inext ;
                                                }
                                   }
                              }
                  }
                  else
                  {

                              /* -------------------------------------------------------------- */
                              /* construct the new element in empty space, Iw [pfree ...] */
                              /* -------------------------------------------------------------- */

                              p = Pe [me] ;
                              pme1 = pfree ;
                              slenme = Len [me] - elenme ;

                              for (knt1 = 1 ; knt1 <= elenme + 1 ; knt1++)
                              {

                                     if (knt1 > elenme)
                                     {
                                                /* search the supervariables in me. */
                                                e = me ;
                                                pj = p ;
                                                ln = slenme ;
                                                AMD_DEBUG2 (("Search sv: "ID" "ID" "ID"\n", me,pj,ln)) ;
                                     }
                                     else
                                     {
                                                /* search the elements in me. */
                                                e = Iw [p++] ;
                                                ASSERT (e >= 0 && e < n) ;
                                                pj = Pe [e] ;
                                                ln = Len [e] ;
                                                AMD_DEBUG2 (("Search element e "ID" in me "ID"\n", e,me)) ;
                                                ASSERT (Elen [e] < EMPTY && W [e] > 0 && pj >= 0) ;
                                     }

                                                     ASSERT (ln >= 0 && (ln == 0 || (pj >= 0 && pj < iwlen))) ;

                                    /* ---------------------------------------------------------
                                     * search for different supervariables and add them to the
                                     * new list, compressing when necessary. this loop is
                                     * executed once for each element in the list and once for
                                     * all the supervariables in the list.
                                     * --------------------------------------------------------- */

                                     for (knt2 = 1 ; knt2 <= ln ; knt2++)
                                     {
                                                i = Iw [pj++] ;
                                                ASSERT (i >= 0 && i < n && (i == me || Elen [i] >=
```

```
EMPTY));
                nvi = Nv [i] ;
                AMD_DEBUG2 ((": "ID" "ID" "ID" "ID"\n", i, Elen [i], Nv [i], wflg)) ;

                if (nvi > 0)
                {

                    /* ------------------------------------------------ */
                    /* compress Iw, if necessary */
                    /* ------------------------------------------------ */

                    if (pfree >= iwlen)
                    {

                        AMD_DEBUG1 (("GARBAGE COLLECTION\n")) ;

                        /* prepare for compressing Iw by adjusting pointers
                         * and lengths so that the lists being searched in
                         * the inner and outer loops contain only the
                         * remaining entries. */

                        Pe [me] = p ;
                        Len [me] -= knt1 ;

                        /* check if nothing left of supervariable me */
                        if (Len [me] == 0) Pe [me] = EMPTY ;
                        Pe [e] = pj ;
                        Len [e] = ln - knt2 ;

                        /* nothing left of element e */
                        if (Len [e] == 0) Pe [e] = EMPTY ;

                        ncmpa++ ;  /* one more garbage collection */

                        /* store first entry of each object in Pe */
                        /* FLIP the first entry in each object */
                        for (j = 0 ; j < n ; j++)
                        {
                            pn = Pe [j] ;
                            if (pn >= 0)
                            {
                                ASSERT (pn >= 0 && pn < iwlen) ;
                                Pe [j] = Iw [pn] ;
                                Iw [pn] = FLIP (j) ;
                            }
                        }

                        /* psrc/pdst point to source/destination */
                        psrc = 0 ;
                        pdst = 0 ;
                        pend = pme1 - 1 ;

                        while (psrc <= pend)
                        {
                            /* search for next FLIP'd entry */
                            j = FLIP (Iw [psrc++]) ;
                            if (j >= 0)
                            {
                                AMD_DEBUG2 (("Got object j: "ID"\n", j)) ;
                                Iw [pdst] = Pe [j] ;
                                Pe [j] = pdst++ ;
                                lenj = Len [j] ;
                                /* copy from source to destination */
                                for (knt3 = 0 ; knt3 <= lenj - 2 ; knt3++)
                                {
                                    Iw [pdst++] = Iw [psrc++] ;
                                }
                            }
                        }

```

```
                                        /* move the new partially-constructed element */
                                        p1 = pdst ;
                                        for (psrc = pme1 ; psrc <= pfree-1 ; psrc++)
                                        {
                                            Iw [pdst++] = Iw [psrc] ;
                                        }
                                        pme1 = p1 ;
                                        pfree = pdst ;
                                        pj = Pe [e] ;
                                        p = Pe [me] ;
                                }

                                /* ------------------------------------------------- */
                                /* i is a principal variable not yet placed in Lme */
                                /* store i in new list */
                                /* ------------------------------------------------- */

                                /* flag i as being in Lme by negating Nv [i] */
                                degme += nvi ;
                                Nv [i] = -nvi ;
                                Iw [pfree++] = i ;
                                AMD_DEBUG2 (("     s: "ID"     nv "ID"\n", i, Nv [i])) ;

                                /* ------------------------------------------------- */
                                /* remove variable i from degree link list */
                                /* ------------------------------------------------- */

                                ilast = Last [i] ;
                                inext = Next [i] ;
                                ASSERT (ilast >= EMPTY && ilast < n) ;
                                ASSERT (inext >= EMPTY && inext < n) ;
                                if (inext != EMPTY) Last [inext] = ilast ;
                                if (ilast != EMPTY)
                                {
                                        Next [ilast] = inext ;
                                }
                                else
                                {
                                        /* i is at the head of the degree list */
                                        ASSERT (Degree [i] >= 0 && Degree [i] < n) ;
                                        Head [Degree [i]] = inext ;
                                }
                            }
                        }

                        if (e != me)
                        {
                                /* set tree pointer and flag to indicate element e is
                                 * absorbed into new element me (the parent of e is me) */
                                AMD_DEBUG1 ((" Element "ID" => "ID"\n", e, me)) ;
                                Pe [e] = FLIP (me) ;
                                W [e] = 0 ;
                        }
                    }

                    pme2 = pfree - 1 ;
            }

            /* ----------------------------------------------------------------- */
            /* me has now been converted into an element in Iw [pme1..pme2] */
            /* ----------------------------------------------------------------- */

            /* degme holds the external degree of new element */
            Degree [me] = degme ;
            Pe [me] = pme1 ;
            Len [me] = pme2 - pme1 + 1 ;
            ASSERT (Pe [me] >= 0 && Pe [me] < iwlen) ;

            Elen [me] = FLIP (nvpiv + degme) ;
            /* FLIP (Elen (me)) is now the degree of pivot (including
             * diagonal part). */

                    #ifndef NDEBUG

                    AMD_DEBUG2 (("New element structure: length= "ID"\n", pme2-pme1+1)) ;
            for (pme = pme1 ; pme <= pme2 ; pme++) AMD_DEBUG3 ((" "ID"", Iw[pme]));
            AMD_DEBUG3 (("\n")) ;

```

```
#endif

        /* --------------------------------------------------------------------- */
        /* make sure that wflg is not too large. */
        /* --------------------------------------------------------------------- */

        /* With the current value of wflg, wflg+n must not cause integer
         * overflow */

        wflg = clear_flag (wflg, wbig, W, n) ;

        /* =========================================================================
 */
        /* COMPUTE (W [e] - wflg) = |Le\Lme| FOR ALL ELEMENTS */
        /* =========================================================================
 */

        /* -----------------------------------------------------------------
         * Scan 1:  compute the external degrees of previous elements with
         * respect to the current element.  That is:
         *       (W [e] - wflg) = |Le \ Lme|
         * for each element e that appears in any supervariable in Lme.  The
         * notation Le refers to the pattern (list of supervariables) of a
         * previous element e, where e is not yet absorbed, stored in
         * Iw [Pe [e] + 1 ... Pe [e] + Len [e]].  The notation Lme
         * refers to the pattern of the current element (stored in
         * Iw [pme1..pme2]).   If aggressive absorption is enabled, and
         * (W [e] - wflg) becomes zero, then the element e will be absorbed
         * in Scan 2.
         * ----------------------------------------------------------------- */

        AMD_DEBUG2 (("me: ")) ;
        for (pme = pme1 ; pme <= pme2 ; pme++)
        {
            i = Iw [pme] ;
            ASSERT (i >= 0 && i < n) ;
            eln = Elen [i] ;
            AMD_DEBUG3 ((""ID" Elen "ID": \n", i, eln)) ;

            if (eln > 0)
            {
                /* note that Nv [i] has been negated to denote i in Lme: */
                nvi = -Nv [i] ;
                ASSERT (nvi > 0 && Pe [i] >= 0 && Pe [i] < iwlen) ;
                wnvi = wflg - nvi ;
                for (p = Pe [i] ; p <= Pe [i] + eln - 1 ; p++)
                {
                    e = Iw [p] ;
                    ASSERT (e >= 0 && e < n) ;
                    we = W [e] ;
                    AMD_DEBUG4 (("    e "ID" we "ID" ", e, we)) ;

                    if (we >= wflg)
                    {
                        /* unabsorbed element e has been seen in this loop */
                        AMD_DEBUG4 (("    unabsorbed, first time seen")) ;
                        we -= nvi ;
                    }
                    else if (we != 0)
                    {
                        /* e is an unabsorbed element */
                        /* this is the first we have seen e in all of Scan 1
 */
                        AMD_DEBUG4 (("    unabsorbed")) ;
                        we = Degree [e] + wnvi ;
                    }

                    AMD_DEBUG4 (("\n")) ;
                    W [e] = we ;
                }
            }
        }

        AMD_DEBUG2 (("\n")) ;

        /* =========================================================================
 */
        /* DEGREE UPDATE AND ELEMENT ABSORPTION */
        /* =========================================================================
 */

        /* -----------------------------------------------------------------
         * Scan 2:  for each i in Lme, sum up the degree of Lme (which is
         * degme), plus the sum of the external degrees of each Le for the
         * elements e appearing within i, plus the supervariables in i.
         * Place i in hash list.
         * ----------------------------------------------------------------- */
```

```
            for (pme = pme1 ; pme <= pme2 ; pme++)
            {
                i = Iw [pme] ;
                ASSERT (i >= 0 && i < n && Nv [i] < 0 && Elen [i] >= 0) ;
                AMD_DEBUG2 (("Updating: i "ID" "ID" "ID"\n", i, Elen[i], Len [i]));
                p1 = Pe [i] ;
                p2 = p1 + Elen [i] - 1 ;
                pn = p1 ;
                hash = 0 ;
                deg = 0 ;
                ASSERT (p1 >= 0 && p1 < iwlen && p2 >= -1 && p2 < iwlen) ;

                /* ------------------------------------------------------------- */
                /* scan the element list associated with supervariable i */
                /* ------------------------------------------------------------- */

                /* UMFPACK/MA38-style approximate degree: */
                if (aggressive)
                {
                    for (p = p1 ; p <= p2 ; p++)
                    {
                        e = Iw [p] ;
                        ASSERT (e >= 0 && e < n) ;
                        we = W [e] ;
                        if (we != 0)
                        {
                            /* e is an unabsorbed element */
                            /* dext = | Le \ Lme | */
                            dext = we - wflg ;
                            if (dext > 0)
                            {
                                deg += dext ;
                                Iw [pn++] = e ;
                                hash += e ;
                                AMD_DEBUG4 ((" e: "ID" hash = "ID"\n",e,hash)) ;
                            }
                            else
                            {
                                /* external degree of e is zero, absorb e into me*/
                                AMD_DEBUG1 ((" Element "ID" =>"ID" (aggressive)\n", e, me)) ;
                                ASSERT (dext == 0) ;
                                Pe [e] = FLIP (me) ;
                                W [e] = 0 ;
                            }
                        }
                    }
                }
                else
                {
                    for (p = p1 ; p <= p2 ; p++)
                    {
                        e = Iw [p] ;
                        ASSERT (e >= 0 && e < n) ;
                        we = W [e] ;
                        if (we != 0)
                        {
                            /* e is an unabsorbed element */
                            dext = we - wflg ;
                            ASSERT (dext >= 0) ;
                            deg += dext ;
                            Iw [pn++] = e ;
                            hash += e ;
                            AMD_DEBUG4 (("   e: "ID" hash = "ID"\n",e,hash)) ;
                        }
                    }
                }

                /* count the number of elements in i (including me): */
                Elen [i] = pn - p1 + 1 ;

                /* ------------------------------------------------------------- */
                /* scan the supervariables in the list associated with i */
                /* ------------------------------------------------------------- */

                /* The bulk of the AMD run time is typically spent in this loop,
                * particularly if the matrix has many dense rows that are not
                * removed prior to ordering. */
                p3 = pn ;
                p4 = p1 + Len [i] ;
                for (p = p2 + 1 ; p < p4 ; p++)
                {
                    j = Iw [p] ;
                    ASSERT (j >= 0 && j < n) ;
```

```
                        nvj = Nv [j] ;
                        if (nvj > 0)
                        {
                                /* j is unabsorbed, and not in Lme. */
                                /* add to degree and add to new list */
                                deg += nvj ;
                                Iw [pn++] = j ;
                                hash += j ;
                                AMD_DEBUG4 (("  s: "ID" hash "ID" Nv[j]= "ID"\n", j,
hash, nvj)) ;
                        }
                }

                /* ------------------------------------------------------------ */
                /* update the degree and check for mass elimination */
                /* ------------------------------------------------------------ */

                /* with aggressive absorption, deg==0 is identical to the
                * Elen [i] == 1 && p3 == pn test, below. */

                        ASSERT (IMPLIES (aggressive, (deg==0) == (Elen[i]==1 && p3==pn)))
;

                if (Elen [i] == 1 && p3 == pn)
                {

                        /* --------------------------------------------------------- */
                        /* mass elimination */
                        /* --------------------------------------------------------- */

                        /* There is nothing left of this node except for an edge to
                        * the current pivot element.  Elen [i] is 1, and there are
                        * no variables adjacent to node i.  Absorb i into the
                        * current pivot element, me.  Note that if there are two or
                        * more mass eliminations, fillin due to mass elimination is
                        * possible within the nvpiv-by-nvpiv pivot block.  It is this
                        * step that causes AMD's analysis to be an upper bound.
                        *
                        * The reason is that the selected pivot has a lower
                        * approximate degree than the true degree of the two mass
                        * eliminated nodes.  There is no edge between the two mass
                        * eliminated nodes.  They are merged with the current pivot
                        * anyway.
                        *
                        * No fillin occurs in the Schur complement, in any case,
                        * and this effect does not decrease the quality of the
                        * ordering itself, just the quality of the nonzero and
                        * flop count analysis.  It also means that the post-ordering
                        * is not an exact elimination tree post-ordering. */

                        AMD_DEBUG1 (("  MASS i "ID" => parent e "ID"\n", i, me)) ;
                        Pe [i] = FLIP (me) ;
                        nvi = -Nv [i] ;
                        degme -= nvi ;
                        nvpiv += nvi ;
                        nel += nvi ;
                        Nv [i] = 0 ;
                        Elen [i] = EMPTY ;
                  }
                else
                {
                                        /*
---------------------------------------------------------- */
                        /* update the upper-bound degree of i */
                        /* --------------------------------------------------------- */

                        /* the following degree does not yet include the size
                        * of the current element, which is added later: */

                        Degree [i] = MIN (Degree [i], deg) ;

                        /* --------------------------------------------------------- */
                        /* add me to the list for i */
                        /* --------------------------------------------------------- */

                        /* move first supervariable to end of list */
                        Iw [pn] = Iw [p3] ;

                                        /* move first element to end of element part of
list */
                        Iw [p3] = Iw [p1] ;

                                        /* add new element, me, to front of list. */
                        Iw [p1] = me ;

                                        /* store the new length of the list in Len [i] */
                        Len [i] = pn - p1 + 1 ;
```

```
                                        /* --------------------------------------------------------- */
                                        /* place in hash bucket.  Save hash key of i in Last [i]. */
                                        /* --------------------------------------------------------- */

                                        /* NOTE: this can fail if hash is negative, because the ANSI C
                                        * standard does not define a % b when a and/or b are negative.
                                        * That's why hash is defined as an unsigned Int, to avoid this
                                        * problem. */
                                        hash = hash % n ;
                                        ASSERT (((Int) hash) >= 0 && ((Int) hash) < n) ;

                                        /* if the Hhead array is not used: */
                                        j = Head [hash] ;
                                        if (j <= EMPTY)
                                        {
                                                    /* degree list is empty, hash head is FLIP (j) */
                                                    Next [i] = FLIP (j) ;
                                                    Head [hash] = FLIP (i) ;
                                        }
                                        else
                                        {
                                                    /* degree list is not empty, use Last [Head [hash]] as
                                                    * hash head. */
                                                    Next [i] = Last [j] ;
                                                    Last [j] = i ;
                                        }

                                        /* if a separate Hhead array is used: *
                                        Next [i] = Hhead [hash] ;
                                        Hhead [hash] = i ;
                                        */

                                        Last [i] = hash ;
                              }
                    }

                    Degree [me] = degme ;

                    /* ----------------------------------------------------------------- */
                    /* Clear the counter array, W [...], by incrementing wflg. */
                    /* ----------------------------------------------------------------- */

                    /* make sure that wflg+n does not cause integer overflow */
                    lemax =  MAX (lemax, degme) ;
                    wflg += lemax ;
                    wflg = clear_flag (wflg, wbig, W, n) ;
                    /*  at this point, W [0..n-1] < wflg holds */

                            /* ========================================================================= */
                            /* SUPERVARIABLE DETECTION */
                            /* ========================================================================= */

                    AMD_DEBUG1 (("Detecting supervariables:\n")) ;
                    for (pme = pme1 ; pme <= pme2 ; pme++)
                    {
                              i = Iw [pme] ;
                              ASSERT (i >= 0 && i < n) ;
                              AMD_DEBUG2 (("Consider i "ID" nv "ID"\n", i, Nv [i])) ;
                              if (Nv [i] < 0)
                              {
                                        /* i is a principal variable in Lme */

                                        /* ---------------------------------------------------------
                                        * examine all hash buckets with 2 or more variables.  We do
                                        * this by examing all unique hash keys for supervariables in
                                        * the pattern Lme of the current element, me
                                        * --------------------------------------------------------- */

                                        /* let i = head of hash bucket, and empty the hash bucket */
                                        ASSERT (Last [i] >= 0 && Last [i] < n) ;
                                        hash = Last [i] ;

                                        /* if Hhead array is not used: */
                                        j = Head [hash] ;
                                        if (j == EMPTY)
                                        {
                                                    /* hash bucket and degree list are both empty */
                                                    i = EMPTY ;
                                        }
                                        else if (j < EMPTY)
                                        {
                                                    /* degree list is empty */
                                                    i = FLIP (j) ;
                                                    Head [hash] = EMPTY ;
```

```
                                        }
                                        else
                                        {
                                                /* degree list is not empty, restore Last [j] of head j */
                                                i = Last [j] ;
                                                Last [j] = EMPTY ;
                                        }

                                        /* if separate Hhead array is used: *
                                        i = Hhead [hash] ;
                                        Hhead [hash] = EMPTY ;
                                        */

                                        ASSERT (i >= EMPTY && i < n) ;
                                        AMD_DEBUG2 (("----i "ID" hash "ID"\n", i, hash)) ;

                                        while (i != EMPTY && Next [i] != EMPTY)
                                        {

                                                /* -----------------------------------------------------
                                                * this bucket has one or more variables following i.
                                                * scan all of them to see if i can absorb any entries
                                                * that follow i in hash bucket.  Scatter i into w.
                                                * ---------------------------------------------------- */

                                                ln = Len [i] ;
                                                eln = Elen [i] ;
                                                ASSERT (ln >= 0 && eln >= 0) ;
                                                ASSERT (Pe [i] >= 0 && Pe [i] < iwlen) ;
                                                                /* do not flag the first element in the list (me) */
                                                for (p = Pe [i] + 1 ; p <= Pe [i] + ln - 1 ; p++)
                                                {
                                                   ASSERT (Iw [p] >= 0 && Iw [p] < n) ;
                                                   W [Iw [p]] = wflg ;
                                                }

                                                /* ----------------------------------------------------- */
                                                /* scan every other entry j following i in bucket */
                                                /* ----------------------------------------------------- */

                                                jlast = i ;
                                                j = Next [i] ;
                                                ASSERT (j >= EMPTY && j < n) ;

                                                while (j != EMPTY)
                                                {
                                                   /* ------------------------------------------------- */
                                                   /* check if j and i have identical nonzero pattern */
                                                   /* ------------------------------------------------- */

                                                   AMD_DEBUG3 (("compare i "ID" and j "ID"\n", i,j)) ;

                                                   /* check if i and j have the same Len and Elen */
                                                   ASSERT (Len [j] >= 0 && Elen [j] >= 0) ;
                                                   ASSERT (Pe [j] >= 0 && Pe [j] < iwlen) ;
                                                   ok = (Len [j] == ln) && (Elen [j] == eln) ;
                                                                   /* skip the first element in the list (me) */
                                                   for (p = Pe [j] + 1 ; ok && p <= Pe [j] + ln - 1 ; p++)
                                                   {
                                                           ASSERT (Iw [p] >= 0 && Iw [p] < n) ;
                                                           if (W [Iw [p]] != wflg) ok = 0 ;
                                                   }

                                                                   if (ok)
                                                   {
                                                           /* --------------------------------------------- */
                                                           /* found it!  j can be absorbed into i */
                                                           /* --------------------------------------------- */

                                                           AMD_DEBUG1 (("found it! j "ID" => i "ID"\n", j,i));
                                                           Pe [j] = FLIP (i) ;

                                                                                   /* both Nv [i]
```

```
and Nv [j] are negated since they */
                                                  /* are in Lme, and the absolute values of
each */
                                                  /* are the number of variables in i and
j: */
                                                  Nv [i] += Nv [j] ;
                                                  Nv [j] = 0 ;
                                                  Elen [j] = EMPTY ;

                                                                      /* delete j
from hash bucket */
                                                  ASSERT (j != Next [j]) ;
                                                  j = Next [j] ;
                                                  Next [jlast] = j ;
                                                            }
                                        else
                                        {
                                                  /* j cannot be absorbed into i */
                                                  jlast = j ;
                                                  ASSERT (j != Next [j]) ;
                                                  j = Next [j] ;
                                        }
                                        ASSERT (j >= EMPTY && j < n) ;
                                      }

                                      /* -------------------------------------------------------
                                      * no more variables can be absorbed into i
                                      * go to next i in bucket and clear flag array
                                      * -------------------------------------------------------
*/

                                      wflg++ ;
                                      i = Next [i] ;
                                      ASSERT (i >= EMPTY && i < n) ;
                                          }
                         }
               }

                    AMD_DEBUG2 (("detect done\n")) ;

                    /* ==========================================================================
*/
                    /* RESTORE DEGREE LISTS AND REMOVE NONPRINCIPAL SUPERVARIABLES FROM ELEMENT */
                    /* ==========================================================================
*/

              p = pme1 ;
              nleft = n - nel ;
              for (pme = pme1 ; pme <= pme2 ; pme++)
              {
                       i = Iw [pme] ;
                       ASSERT (i >= 0 && i < n) ;
                       nvi = -Nv [i] ;
                       AMD_DEBUG3 (("Restore i "ID" "ID"\n", i, nvi)) ;
                       if (nvi > 0)
                       {
                            /* i is a principal variable in Lme */
                            /* restore Nv [i] to signify that i is principal */
                            Nv [i] = nvi ;

                            /* --------------------------------------------------------- */
                            /* compute the external degree (add size of current element) */
                            /* --------------------------------------------------------- */

                            deg = Degree [i] + degme - nvi ;
                            deg = MIN (deg, nleft - nvi) ;
                            ASSERT (IMPLIES (aggressive, deg > 0) && deg >= 0 && deg < n) ;

                            /* --------------------------------------------------------- */
                            /* place the supervariable at the head of the degree list */
                            /* --------------------------------------------------------- */

                            inext = Head [deg] ;
                            ASSERT (inext >= EMPTY && inext < n) ;
                            if (inext != EMPTY) Last [inext] = i ;
                            Next [i] = inext ;
                            Last [i] = EMPTY ;
                            Head [deg] = i ;

                            /* --------------------------------------------------------- */
                            /* save the new degree, and find the minimum degree */
                            /* --------------------------------------------------------- */

                            mindeg = MIN (mindeg, deg) ;
                            Degree [i] = deg ;

                            /* --------------------------------------------------------- */
```

```
                                /* place the supervariable in the element pattern */
                                /* ---------------------------------------------------------- */

                                Iw [p++] = i ;
                                  }
                  }
                  AMD_DEBUG2 (("restore done\n")) ;

                          /* ===================================================================== */
                          /* FINALIZE THE NEW ELEMENT */
                          /* ===================================================================== */

                      AMD_DEBUG2 (("ME = "ID" DONE\n", me)) ;
                  Nv [me] = nvpiv ;

                          /* save the length of the list for the new element me */
                  Len [me] = p - pme1 ;
                  if (Len [me] == 0)
                  {
                             /* there is nothing left of the current pivot element */
                             /* it is a root of the assembly tree */
                             Pe [me] = EMPTY ;
                             W [me] = 0 ;
                  }

                          if (elenme != 0)
                  {
                             /* element was not constructed in place: deallocate part of */
                             /* it since newly nonprincipal variables may have been removed */
                             pfree = p ;
                  }

                  /* The new element has nvpiv pivots and the size of the contribution
                  * block for a multifrontal method is degme-by-degme, not including
                  * the "dense" rows/columns.  If the "dense" rows/columns are included,
                  * the frontal matrix is no larger than
                  * (degme+ndense)-by-(degme+ndense).
                  */

                  if (ABIPInfo != (abip_float *) ABIP_NULL)
                  {
                             f = nvpiv ;
                             r = degme + ndense ;
                             dmax = MAX (dmax, f + r) ;

                             /* number of nonzeros in L (excluding the diagonal) */
                             lnzme = f*r + (f-1)*f/2 ;
                             lnz += lnzme ;

                             /* number of divide operations for LDL' and for LU */
                             ndiv += lnzme ;

                             /* number of multiply-subtract pairs for LU */
                             s = f*r*r + r*(f-1)*f + (f-1)*f*(2*f-1)/6 ;
                             nms_lu += s ;

                             /* number of multiply-subtract pairs for LDL' */
                             nms_ldl += (s + lnzme)/2 ;
                  }

                          #ifndef NDEBUG

                          AMD_DEBUG2 (("finalize done nel "ID" n "ID"\n   ::::\n", nel, n)) ;
                  for (pme = Pe [me] ; pme <= Pe [me] + Len [me] - 1 ; pme++)
                  {
                             AMD_DEBUG3 ((" "ID"", Iw [pme])) ;
                  }
                  AMD_DEBUG3 (("\n")) ;

                          #endif
              }

              /* ===================================================================== */
              /* DONE SELECTING PIVOTS */
              /* ===================================================================== */

              if (ABIPInfo != (abip_float *) ABIP_NULL)
              {

                  /* count the work to factorize the ndense-by-ndense submatrix */
                  f = ndense ;
                  dmax = MAX (dmax, (abip_float) ndense) ;

                  /* number of nonzeros in L (excluding the diagonal) */
                  lnzme = (f-1)*f/2 ;
```

```
            lnz += lnzme ;

            /* number of divide operations for LDL' and for LU */
            ndiv += lnzme ;

            /* number of multiply-subtract pairs for LU */
            s = (f-1)*f*(2*f-1)/6 ;
            nms_lu += s ;

            /* number of multiply-subtract pairs for LDL' */
            nms_ldl += (s + lnzme)/2 ;

            /* number of nz's in L (excl. diagonal) */
            ABIPInfo [AMD_LNZ] = lnz ;

            /* number of divide ops for LU and LDL' */
            ABIPInfo [AMD_NDIV] = ndiv ;

            /* number of multiply-subtract pairs for LDL' */
            ABIPInfo [AMD_NMULTSUBS_LDL] = nms_ldl ;

            /* number of multiply-subtract pairs for LU */
            ABIPInfo [AMD_NMULTSUBS_LU] = nms_lu ;

            /* number of "dense" rows/columns */
            ABIPInfo [AMD_NDENSE] = ndense ;

            /* largest front is dmax-by-dmax */
            ABIPInfo [AMD_DMAX] = dmax ;

            /* number of garbage collections in AMD */
            ABIPInfo [AMD_NCMPA] = ncmpa ;

            /* successful ordering */
            ABIPInfo [AMD_STATUS] = AMD_OK ;
         }

         /* ========================================================================= */
         /* POST-ORDERING */
         /* ========================================================================= */

         /* -------------------------------------------------------------------------
         * Variables at this point:
         *
         * Pe: holds the elimination tree.  The parent of j is FLIP (Pe [j]),
         *   or EMPTY if j is a root.  The tree holds both elements and
         *   non-principal (unordered) variables absorbed into them.
         *   Dense variables are non-principal and unordered.
         *
         * Elen: holds the size of each element, including the diagonal part.
         *   FLIP (Elen [e]) > 0 if e is an element.  For unordered
         *   variables i, Elen [i] is EMPTY.
         *
         * Nv: Nv [e] > 0 is the number of pivots represented by the element e.
         *   For unordered variables i, Nv [i] is zero.
         *
         * Contents no longer needed:
         *   W, Iw, Len, Degree, Head, Next, Last.
         *
         * The matrix itself has been destroyed.
         *
         * n: the size of the matrix.
         * No other scalars needed (pfree, iwlen, etc.)
         * ------------------------------------------------------------------------- */

         /* restore Pe */
         for (i = 0 ; i < n ; i++)
         {
            Pe [i] = FLIP (Pe [i]) ;
         }

         /* restore Elen, for output information, and for postordering */
         for (i = 0 ; i < n ; i++)
         {
            Elen [i] = FLIP (Elen [i]) ;
         }

         /* Now the parent of j is Pe [j], or EMPTY if j is a root.  Elen [e] > 0
         * is the size of element e.  Elen [i] is EMPTY for unordered variable i. */

         #ifndef NDEBUG

         AMD_DEBUG2 (("\nTree:\n")) ;
         for (i = 0 ; i < n ; i++)
         {
            AMD_DEBUG2 ((" "ID" parent: "ID"   ", i, Pe [i])) ;
            ASSERT (Pe [i] >= EMPTY && Pe [i] < n) ;
```

```
        if (Nv [i] > 0)
        {
            /* this is an element */
            e = i ;
            AMD_DEBUG2 ((" element, size is "ID"\n", Elen [i])) ;
            ASSERT (Elen [e] > 0) ;
        }
        AMD_DEBUG2 (("\n")) ;
    }

    AMD_DEBUG2 (("\nelements:\n")) ;

    for (e = 0 ; e < n ; e++)
    {
        if (Nv [e] > 0)
        {
            AMD_DEBUG3 (("Element e= "ID" size "ID" nv "ID" \n", e, Elen [e], Nv [e]))
;
        }
    }

    AMD_DEBUG2 (("\nvariables:\n")) ;
    for (i = 0 ; i < n ; i++)
    {
        Int cnt ;
        if (Nv [i] == 0)
        {
            AMD_DEBUG3 (("i unordered: "ID"\n", i)) ;
            j = Pe [i] ;
            cnt = 0 ;
            AMD_DEBUG3 (("  j: "ID"\n", j)) ;
            if (j == EMPTY)
            {
                AMD_DEBUG3 (("  i is a dense variable\n")) ;
            }
            else
            {
                ASSERT (j >= 0 && j < n) ;
                while (Nv [j] == 0)
                {
                    AMD_DEBUG3 (("  j : "ID"\n", j)) ;
                    j = Pe [j] ;
                    AMD_DEBUG3 (("  j:: "ID"\n", j)) ;
                    cnt++ ;
                    if (cnt > n) break ;
                }
                e = j ;
                AMD_DEBUG3 (("  got to e: "ID"\n", e)) ;
            }
        }
    }

#endif

    /* ========================================================================= */
    /* compress the paths of the variables */
    /* ========================================================================= */

    for (i = 0 ; i < n ; i++)
    {
        if (Nv [i] == 0)
        {

            /* -------------------------------------------------------------
             * i is an un-ordered row.  Traverse the tree from i until
             * reaching an element, e.  The element, e, was the principal
             * supervariable of i and all nodes in the path from i to when e
             * was selected as pivot.
             * ------------------------------------------------------------- */

            AMD_DEBUG1 (("Path compression, i unordered: "ID"\n", i)) ;
            j = Pe [i] ;
            ASSERT (j >= EMPTY && j < n) ;
            AMD_DEBUG3 ((" j: "ID"\n", j)) ;

                if (j == EMPTY)
            {
                /* Skip a dense variable.  It has no parent. */
                AMD_DEBUG3 (("      i is a dense variable\n")) ;
                continue ;
            }

            /* while (j is a variable) */
            while (Nv [j] == 0)
            {
                AMD_DEBUG3 (("      j : "ID"\n", j)) ;
                j = Pe [j] ;
```

```
                                AMD_DEBUG3 (("      j:: "ID"\n", j)) ;
                                ASSERT (j >= 0 && j < n) ;
                        }

                                  /* got to an element e */
                        e = j ;
                        AMD_DEBUG3 (("got to e: "ID"\n", e)) ;

                        /* -------------------------------------------------------------
                        * traverse the path again from i to e, and compress the path
                        * (all nodes point to e).  Path compression allows this code to
                        * compute in O(n) time.
                        * ------------------------------------------------------------- */

                        j = i ;

                                  /* while (j is a variable) */
                        while (Nv [j] == 0)
                        {
                                jnext = Pe [j] ;
                                AMD_DEBUG3 (("j "ID" jnext "ID"\n", j, jnext)) ;
                                Pe [j] = e ;
                                j = jnext ;
                                ASSERT (j >= 0 && j < n) ;
                        }
            }
        }

        /* ========================================================================= */
        /* postorder the assembly tree */
        /* ========================================================================= */

        AMD_postorder (n, Pe, Nv, Elen,
           W,            /* output order */
           Head, Next, Last) ;  /* workspace */

        /* ========================================================================= */
        /* compute output permutation and inverse permutation */
        /* ========================================================================= */

        /* W [e] = k means that element e is the kth element in the new
        * order.  e is in the range 0 to n-1, and k is in the range 0 to
        * the number of elements.  Use Head for inverse order. */

        for (k = 0 ; k < n ; k++)
        {
           Head [k] = EMPTY ;
           Next [k] = EMPTY ;
        }

        for (e = 0 ; e < n ; e++)
        {
           k = W [e] ;
           ASSERT ((k == EMPTY) == (Nv [e] == 0)) ;
           if (k != EMPTY)
           {
                        ASSERT (k >= 0 && k < n) ;
                        Head [k] = e ;
           }
        }

        /* construct output inverse permutation in Next,
        * and permutation in Last */
        nel = 0 ;

        for (k = 0 ; k < n ; k++)
        {
           e = Head [k] ;
           if (e == EMPTY) break ;
           ASSERT (e >= 0 && e < n && Nv [e] > 0) ;
           Next [e] = nel ;
           nel += Nv [e] ;
        }
        ASSERT (nel == n - ndense) ;

        /* order non-principal variables (dense, & those merged into supervar's) */
        for (i = 0 ; i < n ; i++)
        {
           if (Nv [i] == 0)
           {
                        e = Pe [i] ;
                        ASSERT (e >= EMPTY && e < n) ;

                                  if (e != EMPTY)
                        {
                                /* This is an unordered variable that was merged
                                * into element e via supernode detection or mass
```

```
                                * elimination of i when e became the pivot element.
                                * Place i in order just before e. */
                                ASSERT (Next [i] == EMPTY && Nv [e] > 0) ;
                                Next [i] = Next [e] ;
                                Next [e]++ ;
                            }
                            else
                            {
                                /* This is a dense unordered variable, with no parent.
                                * Place it last in the output order. */
                                Next [i] = nel++ ;
                            }
                }
            }

            ASSERT (nel == n) ;

            AMD_DEBUG2 (("\n\nPerm:\n")) ;

            for (i = 0 ; i < n ; i++)
            {
                k = Next [i] ;
                ASSERT (k >= 0 && k < n) ;
                Last [k] = i ;
                AMD_DEBUG2 (("   perm ["ID"] = "ID"\n", k, i)) ;
            }
}
```

5.11 external/amd/amd_aat.c File Reference

```
#include "amd_internal.h"
```

Functions

- GLOBAL size_t AMD_aat (Int n, const Int Ap[ ], const Int Ai[ ], Int Len[ ], Int Tp[ ], abip_float ABIPInfo[ ])

5.11.1 Function Documentation

5.11.1.1 AMD_aat()

```
GLOBAL size_t AMD_aat (
            Int n,
            const Int Ap[ ],
            const Int Ai[ ],
            Int Len[ ],
            Int Tp[ ],
            abip_float ABIPInfo[ ] )
```

Definition at line 20 of file amd_aat.c.

5.12 amd_aat.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_aat ============================================================= */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                     */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* AMD_aat:  compute the symmetry of the pattern of A, and count the number of
 * nonzeros each column of A+A' (excluding the diagonal).  Assumes the input
 * matrix has no errors, with sorted columns and no duplicates
 * (AMD_valid (n, n, Ap, Ai) must be AMD_OK, but this condition is not
 * checked).
 */

#include "amd_internal.h"

GLOBAL size_t AMD_aat  /* returns nz in A+A' */
(
        Int n,
        const Int Ap [ ],
        const Int Ai [ ],
        Int Len [ ],     /* Len [j]: length of column j of A+A', excl diagonal*/
        Int Tp [ ],      /* workspace of size n */
        abip_float ABIPInfo [ ]
)
{
        Int p1;
        Int p2;
        Int p;
        Int pj;
        Int pj2;

        Int i;
        Int j;
        Int k;

        Int nzdiag;
        Int nzboth;
        Int nz;

        abip_float sym ;
        size_t nzaat ;

    #ifndef NDEBUG

        AMD_debug_init ("AMD AAT") ;
        for (k = 0 ; k < n ; k++) Tp [k] = EMPTY ;
        ASSERT (AMD_valid (n, n, Ap, Ai) == AMD_OK) ;

    #endif

        if (ABIPInfo != (abip_float *) ABIP_NULL)
        {
        /* clear the ABIPInfo array, if it exists */
        for (i = 0 ; i < AMD_INFO ; i++)
        {
                ABIPInfo [i] = EMPTY ;
        }
        ABIPInfo [AMD_STATUS] = AMD_OK ;
        }

        for (k = 0 ; k < n ; k++)
        {
        Len [k] = 0 ;
        }

        nzdiag = 0 ;
        nzboth = 0 ;
        nz = Ap [n] ;

        for (k = 0 ; k < n ; k++)
        {
        p1 = Ap [k] ;
        p2 = Ap [k+1] ;
        AMD_DEBUG2 (("\nAAT Column: "ID" p1: "ID" p2: "ID"\n", k, p1, p2)) ;

        /* construct A+A' */
        for (p = p1 ; p < p2 ; )
        {
```

```
        /* scan the upper triangular part of A */
        j = Ai [p] ;

        if (j < k)
        {
        /* entry A (j,k) is in the strictly upper triangular part,
        * add both A (j,k) and A (k,j) to the matrix A+A' */
        Len [j]++ ;
        Len [k]++ ;
        AMD_DEBUG3 (("    upper ("ID","ID") ("ID","ID")\n", j,k, k,j));
        p++ ;
        }
        else if (j == k)
        {
        /* skip the diagonal */
        p++ ;
        nzdiag++ ;
        break ;
        }
        else /* j > k */
        {
        /* first entry below the diagonal */
        break ;
        }

        /* scan lower triangular part of A, in column j until reaching
        * row k.  Start where last scan left off. */
        ASSERT (Tp [j] != EMPTY) ;
        ASSERT (Ap [j] <= Tp [j] && Tp [j] <= Ap [j+1]) ;
        pj2 = Ap [j+1] ;

        for (pj = Tp [j] ; pj < pj2 ; )
        {
        i = Ai [pj] ;

        if (i < k)
        {
                /* A (i,j) is only in the lower part, not in upper.
                * add both A (i,j) and A (j,i) to the matrix A+A' */
                Len [i]++ ;
                Len [j]++ ;
                AMD_DEBUG3 (("    lower ("ID","ID") ("ID","ID")\n", i,j, j,i)) ;
                pj++ ;
        }
        else if (i == k)
        {
                /* entry A (k,j) in lower part and A (j,k) in upper */
                pj++ ;
                nzboth++ ;
                break ;
        }
        else /* i > k */
        {
                /* consider this entry later, when k advances to i */
                break ;
        }
        }
        Tp [j] = pj ;
    }

    /* Tp [k] points to the entry just below the diagonal in column k */
    Tp [k] = p ;
    }

    /* clean up, for remaining mismatched entries */
    for (j = 0 ; j < n ; j++)
    {
    for (pj = Tp [j] ; pj < Ap [j+1] ; pj++)
    {
        i = Ai [pj] ;

        /* A (i,j) is only in the lower part, not in upper.
        * add both A (i,j) and A (j,i) to the matrix A+A' */
        Len [i]++ ;
        Len [j]++ ;
        AMD_DEBUG3 (("    lower cleanup ("ID","ID") ("ID","ID")\n", i,j, j,i)) ;
    }
    }

    /* --------------------------------------------------------------------- */
    /* compute the symmetry of the nonzero pattern of A */
    /* --------------------------------------------------------------------- */

    /* Given a matrix A, the symmetry of A is:
    *   B = tril (spones (A), -1) + triu (spones (A), 1) ;
    *  sym = nnz (B & B') / nnz (B) ;
    *  or 1 if nnz (B) is zero. */
```

```
        if (nz == nzdiag)
        {
        sym = 1 ;
        }
        else
        {
        sym = (2 * (abip_float) nzboth) / ((abip_float) (nz - nzdiag)) ;
        }

        nzaat = 0 ;
        for (k = 0 ; k < n ; k++)
        {
        nzaat += Len [k] ;
        }

        AMD_DEBUG1 (("AMD nz in A+A', excluding diagonal (nzaat) = %g\n", (abip_float) nzaat)) ;
        AMD_DEBUG1 (("   nzboth: "ID" nz: "ID" nzdiag: "ID" symmetry: %g\n", nzboth, nz, nzdiag, sym))
    ;

        if (ABIPInfo != (abip_float *) ABIP_NULL)
        {
        ABIPInfo [AMD_STATUS] = AMD_OK ;
        ABIPInfo [AMD_N] = n ;
        ABIPInfo [AMD_NZ] = nz ;
        ABIPInfo [AMD_SYMMETRY] = sym ;         /* symmetry of pattern of A */
        ABIPInfo [AMD_NZDIAG] = nzdiag ;          /* nonzeros on diagonal of A */
        ABIPInfo [AMD_NZ_A_PLUS_AT] = nzaat ;       /* nonzeros in A+A' */
        }

        return (nzaat) ;
}
```

5.13 external/amd/amd_control.c File Reference

`#include "amd_internal.h"`

Functions

- GLOBAL void AMD_control (abip_float Control[ ])

5.13.1 Function Documentation

5.13.1.1 AMD_control()

```
GLOBAL void AMD_control (
            abip_float Control[ ] )
```

Definition at line 18 of file amd_control.c.

5.14 amd_control.c

Go to the documentation of this file.

```c
/* ========================================================================= */
/* === AMD_control ========================================================= */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                     */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* User-callable.  Prints the control parameters for AMD.  See amd.h
 * for details.  If the Control array is not present, the defaults are
 * printed instead.
 */

#include "amd_internal.h"

GLOBAL void AMD_control
(
            abip_float Control [ ]
)
{
            abip_float alpha ;
            Int aggressive ;

            if (Control != (abip_float *) ABIP_NULL)
            {
               alpha = Control [AMD_DENSE] ;
               aggressive = Control [AMD_AGGRESSIVE] != 0 ;
            }
            else
            {
               alpha = AMD_DEFAULT_DENSE ;
               aggressive = AMD_DEFAULT_AGGRESSIVE ;
            }

            PRINTF (("\nAMD version %d.%d.%d, %s: approximate minimum degree ordering\n"
               "    dense row parameter: %g\n", AMD_MAIN_VERSION, AMD_SUB_VERSION, AMD_SUBSUB_VERSION,
 AMD_DATE, alpha)) ;

            if (alpha < 0)
            {
               PRINTF (("    no rows treated as dense\n")) ;
            }
            else
            {
               PRINTF ((
                            "    (rows with more than max (%g * sqrt (n), 16) entries are\n"
                            "    considered \"dense\", and placed last in output permutation)\n",
 alpha)) ;
            }

            if (aggressive)
            {
               PRINTF (("    aggressive absorption:  yes\n")) ;
            }
            else
            {
               PRINTF (("    aggressive absorption:  no\n")) ;
            }

            PRINTF (("    size of AMD integer: %d\n\n", sizeof (Int))) ;
}
```

5.15 external/amd/amd_defaults.c File Reference

`#include "amd_internal.h"`

Functions

- GLOBAL void AMD_defaults (abip_float Control[ ])

5.15.1 Function Documentation

5.15.1.1 AMD_defaults()

```
GLOBAL void AMD_defaults (
            abip_float Control[ ] )
```

Definition at line 21 of file amd_defaults.c.

5.16 amd_defaults.c

Go to the documentation of this file.

```
00001 /* ========================================================================= */
00002 /* === AMD_defaults ======================================================== */
00003 /* ========================================================================= */
00004
00005 /* ------------------------------------------------------------------------- */
00006 /* AMD, Copyright (c) Timothy A. Davis,                          */
00007 /* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
00008 /* email: DrTimothyAldenDavis@gmail.com                                      */
00009 /* ------------------------------------------------------------------------- */
00010
00011 /* User-callable.  Sets default control parameters for AMD.  See amd.h
00012  * for details.
00013  */
00014
00015 #include "amd_internal.h"
00016
00017 /* ========================================================================= */
00018 /* === AMD defaults ======================================================== */
00019 /* ========================================================================= */
00020
00021 GLOBAL void AMD_defaults
00022 (
00023         abip_float Control [ ]
00024 )
00025 {
00026         Int i;
00027
00028         if (Control != (abip_float *) ABIP_NULL)
00029         {
00030         for (i = 0 ; i < AMD_CONTROL ; i++)
00031         {
00032                 Control [i] = 0 ;
00033         }
00034         Control [AMD_DENSE] = AMD_DEFAULT_DENSE ;
00035         Control [AMD_AGGRESSIVE] = AMD_DEFAULT_AGGRESSIVE ;
00036         }
00037 }
```

5.17 external/amd/amd_dump.c File Reference

```
#include "amd_internal.h"
```

Functions

- GLOBAL void AMD_debug_init (char ∗s)
- GLOBAL void AMD_dump (Int n, Int Pe[ ], Int Iw[ ], Int Len[ ], Int iwlen, Int pfree, Int Nv[ ], Int Next[ ], Int Last[ ], Int Head[ ], Int Elen[ ], Int Degree[ ], Int W[ ], Int nel)

Variables

- GLOBAL Int AMD_debug = -999

5.17.1 Function Documentation

5.17.1.1 AMD_debug_init()

```
GLOBAL void AMD_debug_init (
            char * s )
```

Definition at line 29 of file amd_dump.c.

5.17.1.2 AMD_dump()

```
GLOBAL void AMD_dump (
            Int n,
            Int Pe[ ],
            Int Iw[ ],
            Int Len[ ],
            Int iwlen,
            Int pfree,
            Int Nv[ ],
            Int Next[ ],
            Int Last[ ],
            Int Head[ ],
            Int Elen[ ],
            Int Degree[ ],
            Int W[ ],
            Int nel )
```

Definition at line 58 of file amd_dump.c.

5.17.2 Variable Documentation

5.17.2.1 AMD_debug

```
GLOBAL Int AMD_debug = -999
```

Definition at line 21 of file amd_dump.c.

5.18 amd_dump.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_dump ============================================================ */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                     */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* Debugging routines for AMD.  Not used if NDEBUG is not defined at compile-
 * time (the default).  See comments in amd_internal.h on how to enable
 * debugging.  Not user-callable.
 */

#include "amd_internal.h"

#ifndef NDEBUG

/* This global variable is present only when debugging */
GLOBAL Int AMD_debug = -999 ;      /* default is no debug printing */

/* ========================================================================= */
/* === AMD_debug_init ====================================================== */
/* ========================================================================= */

/* Sets the debug print level, by reading the file debug.amd (if it exists) */

GLOBAL void AMD_debug_init ( char *s )
{
        FILE *f ;
        f = fopen ("debug.amd", "r") ;

        if (f == (FILE *) ABIP_NULL)
        {
        AMD_debug = -999 ;
        }
        else
        {
        fscanf (f, ID, &AMD_debug) ;
        fclose (f) ;
        }

        if (AMD_debug >= 0)
        {
        printf ("%s: AMD_debug_init, D= "ID"\n", s, AMD_debug) ;
        }
}

/* ========================================================================= */
/* === AMD_dump ============================================================ */
/* ========================================================================= */

/* Dump AMD's data structure, except for the hash buckets.  This routine
 * cannot be called when the hash buckets are non-empty.
 */

GLOBAL void AMD_dump (
        Int n,                  /* A is n-by-n */
        Int Pe [ ],             /* pe [0..n-1]: index in iw of start of row i */
        Int Iw [ ],             /* workspace of size iwlen, iwlen [0..pfree-1]
                   * holds the matrix on input */
        Int Len [ ],            /* len [0..n-1]: length for row i */
        Int iwlen,              /* length of iw */
        Int pfree,              /* iw [pfree ... iwlen-1] is empty on input */
        Int Nv [ ],             /* nv [0..n-1] */
        Int Next [ ],           /* next [0..n-1] */
        Int Last [ ],           /* last [0..n-1] */
        Int Head [ ],           /* head [0..n-1] */
        Int Elen [ ],           /* size n */
        Int Degree [ ],         /* size n */
        Int W [ ],              /* size n */
        Int nel
)
{
        Int i;
        Int pe;
        Int elen;
        Int nv;
        Int len;
        Int e;
        Int p;
```

```
        Int k;
        Int j;
        Int deg;
        Int w;
        Int cnt;
        Int ilast;

        if (AMD_debug < 0) return ;
        ASSERT (pfree <= iwlen) ;
        AMD_DEBUG3 (("\nAMD dump, pfree: "ID"\n", pfree)) ;
        for (i = 0 ; i < n ; i++)
        {
        pe = Pe [i] ;
        elen = Elen [i] ;
        nv = Nv [i] ;
        len = Len [i] ;
        w = W [i] ;

        if (elen >= EMPTY)
        {
                if (nv == 0)
                {
                AMD_DEBUG3 (("\nI "ID": nonprincipal:    ", i)) ;
                ASSERT (elen == EMPTY) ;

                if (pe == EMPTY)
                {
                        AMD_DEBUG3 ((" dense node\n")) ;
                        ASSERT (w == 1) ;
                }
                else
                {
                        ASSERT (pe < EMPTY) ;
                        AMD_DEBUG3 ((" i "ID" -> parent "ID"\n", i, FLIP (Pe[i])));
                }
                }
                else
                {
                AMD_DEBUG3 (("\nI "ID": active principal supervariable:\n",i));
                AMD_DEBUG3 (("   nv(i): "ID"  Flag: %d\n", nv, (nv < 0))) ;

                ASSERT (elen >= 0) ;
                ASSERT (nv > 0 && pe >= 0) ;
                p = pe ;
                AMD_DEBUG3 (("   e/s: ")) ;

                if (elen == 0) AMD_DEBUG3 ((" : ")) ;
                ASSERT (pe + len <= pfree) ;

                for (k = 0 ; k < len ; k++)
                {
                        j = Iw [p] ;
                        AMD_DEBUG3 (("  "ID"", j)) ;
                        ASSERT (j >= 0 && j < n) ;

                        if (k == elen-1) AMD_DEBUG3 ((" : ")) ;
                        p++ ;
                }

                AMD_DEBUG3 (("\n")) ;
                }
        }
        else
        {
                e = i ;

                if (w == 0)
                {
                AMD_DEBUG3 (("\nE "ID": absorbed element: w "ID"\n", e, w)) ;
                ASSERT (nv > 0 && pe < 0) ;
                AMD_DEBUG3 ((" e "ID" -> parent "ID"\n", e, FLIP (Pe [e]))) ;
                }
                else
                {
                AMD_DEBUG3 (("\nE "ID": unabsorbed element: w "ID"\n", e, w)) ;
                ASSERT (nv > 0 && pe >= 0) ;
                p = pe ;
                AMD_DEBUG3 ((" : ")) ;
                ASSERT (pe + len <= pfree) ;

                for (k = 0 ; k < len ; k++)
                {
                        j = Iw [p] ;
                        AMD_DEBUG3 (("  "ID"", j)) ;
                        ASSERT (j >= 0 && j < n) ;
                        p++ ;
                }
```

```

                AMD_DEBUG3 (("\n")) ;
                }
        }
        }

        /* this routine cannot be called when the hash buckets are non-empty */
        AMD_DEBUG3 (("\nDegree lists:\n")) ;
        if (nel >= 0)
        {
        cnt = 0 ;

        for (deg = 0 ; deg < n ; deg++)
        {
                if (Head [deg] == EMPTY) continue ;
                ilast = EMPTY ;
                AMD_DEBUG3 ((ID": \n", deg)) ;

                for (i = Head [deg] ; i != EMPTY ; i = Next [i])
                {
                AMD_DEBUG3 (("   "ID" : next "ID" last "ID" deg "ID"\n", i, Next [i], Last [i], Degree
    [i])) ;
                ASSERT (i >= 0 && i < n && ilast == Last [i] && deg == Degree [i]) ;
                cnt += Nv [i] ;
                ilast = i ;
                }

                AMD_DEBUG3 (("\n")) ;
        }

        ASSERT (cnt == n - nel) ;
        }
}

#endif
```

5.19 external/amd/amd_global.c File Reference

```
#include <stdlib.h>
#include "glbopts.h"
```

Macros

- #define ABIP_NULL 0

Variables

- void ∗(∗ amd_malloc)(size_t) = malloc
- void(∗ amd_free)(void ∗) = free
- void ∗(∗ amd_realloc)(void ∗, size_t) = realloc
- void ∗(∗ amd_calloc)(size_t, size_t) = calloc
- int(∗ amd_printf)(const char ∗,...) = ABIP_NULL

5.19.1 Macro Definition Documentation

5.19.1.1 ABIP_NULL

```
#define ABIP_NULL 0
```

Definition at line 20 of file amd_global.c.

5.19.2 Variable Documentation

5.19.2.1 amd_calloc

```
void *(* amd_calloc) (size_t, size_t) (
            size_t ,
            size_t  ) = calloc
```

Definition at line 54 of file amd_global.c.

5.19.2.2 amd_free

```
void(* amd_free) (void *) (
            void *  ) = free
```

Definition at line 52 of file amd_global.c.

5.19.2.3 amd_malloc

```
void *(* amd_malloc) (size_t) (
            size_t  ) = malloc
```

Definition at line 51 of file amd_global.c.

5.19.2.4 amd_printf

```
int(* amd_printf) (const char *,...)  (
            const char * ,
             ...  )  = ABIP_NULL
```

Definition at line 75 of file amd_global.c.

5.19.2.5 amd_realloc

```
void *(* amd_realloc) (void *, size_t) (
            void * ,
            size_t  ) = realloc
```

Definition at line 53 of file amd_global.c.

5.20 amd_global.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === amd_global ========================================================== */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                     */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

#include <stdlib.h>
#include "glbopts.h"

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#include "matrix.h"
#endif

#ifndef ABIP_NULL
#define ABIP_NULL 0
#endif

/* ========================================================================= */
/* === Default AMD memory manager ========================================== */
/* ========================================================================= */

/* The user can redefine these global pointers at run-time to change the memory
 * manager used by AMD.  AMD only uses malloc and free; realloc and calloc are
 * include for completeness, in case another package wants to use the same
 * memory manager as AMD.
 *
 * If compiling as a MATLAB mexFunction, the default memory manager is mxMalloc.
 * You can also compile AMD as a standard ANSI-C library and link a mexFunction
 * against it, and then redefine these pointers at run-time, in your
 * mexFunction.
 *
 * If -DNMALLOC is defined at compile-time, no memory manager is specified at
 * compile-time.  You must then define these functions at run-time, before
 * calling AMD, for AMD to work properly.
 */

#ifndef NMALLOC
#ifdef MATLAB_MEX_FILE
/* MATLAB mexFunction: */
void *(*amd_malloc) (size_t) = mxMalloc ;
void (*amd_free) (void *) = mxFree ;
void *(*amd_realloc) (void *, size_t) = mxRealloc ;
void *(*amd_calloc) (size_t, size_t) = mxCalloc ;
#else
/* standard ANSI-C: */
void *(*amd_malloc) (size_t) = malloc ;
void (*amd_free) (void *) = free ;
void *(*amd_realloc) (void *, size_t) = realloc ;
void *(*amd_calloc) (size_t, size_t) = calloc ;
#endif
#else
/* no memory manager defined at compile-time; you MUST define one at run-time */
void *(*amd_malloc) (size_t) = ABIP_NULL ;
void (*amd_free) (void *) = ABIP_NULL ;
void *(*amd_realloc) (void *, size_t) = ABIP_NULL ;
void *(*amd_calloc) (size_t, size_t) = ABIP_NULL ;
#endif

/* ========================================================================= */
/* === Default AMD printf routine ========================================== */
/* ========================================================================= */

/* The user can redefine this global pointer at run-time to change the printf
 * routine used by AMD.  If ABIP_NULL, no printing occurs.
 *
 * If -DNPRINT is defined at compile-time, stdio.h is not included.  Printing
 * can then be enabled at run-time by setting amd_printf to a non-ABIP_NULL function.
 */

int (*amd_printf) (const char *, ...) = ABIP_NULL ;
```

5.21 external/amd/amd_info.c File Reference

```
#include "amd_internal.h"
```

Macros

- #define PRI(format, x) { if (x >= 0) { PRINTF ((format, x)) ; }}

Functions

- GLOBAL void AMD_info (abip_float ABIPInfo[ ])

5.21.1 Macro Definition Documentation

5.21.1.1 PRI

```
#define PRI(
            format,
            x ) { if (x >= 0) { PRINTF ((format, x)) ; }}
```

Definition at line 17 of file amd_info.c.

5.21.2 Function Documentation

5.21.2.1 AMD_info()

```
GLOBAL void AMD_info (
            abip_float ABIPInfo[ ] )
```

Definition at line 19 of file amd_info.c.

5.22 amd_info.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_info ============================================================ */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                      */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* User-callable.  Prints the output statistics for AMD.  See amd.h
 * for details.  If the ABIPInfo array is not present, nothing is printed.
 */

#include "amd_internal.h"

#define PRI(format,x) { if (x >= 0) { PRINTF ((format, x)) ; }}

GLOBAL void AMD_info
(
            abip_float ABIPInfo [ ]
)
{
            abip_float n;
            abip_float ndiv;
            abip_float nmultsubs_ldl;
            abip_float nmultsubs_lu;
            abip_float lnz;
            abip_float lnzd;

            PRINTF (("\nAMD version %d.%d.%d, %s, results:\n", AMD_MAIN_VERSION, AMD_SUB_VERSION, AMD_SUBSUB_VERSION, AMD_DATE)) ;

            if (!ABIPInfo)
            {
               return ;
            }

            n = ABIPInfo [AMD_N] ;
            ndiv = ABIPInfo [AMD_NDIV] ;
            nmultsubs_ldl = ABIPInfo [AMD_NMULTSUBS_LDL] ;
            nmultsubs_lu = ABIPInfo [AMD_NMULTSUBS_LU] ;
            lnz = ABIPInfo [AMD_LNZ] ;
            lnzd = (n >= 0 && lnz >= 0) ? (n + lnz) : (-1) ;

            /* AMD return status */
            PRINTF (("    status: ")) ;
            if (ABIPInfo [AMD_STATUS] == AMD_OK)
            {
               PRINTF (("OK\n")) ;
            }
            else if (ABIPInfo [AMD_STATUS] == AMD_OUT_OF_MEMORY)
            {
               PRINTF (("out of memory\n")) ;
            }
            else if (ABIPInfo [AMD_STATUS] == AMD_INVALID)
            {
               PRINTF (("invalid matrix\n")) ;
            }
            else if (ABIPInfo [AMD_STATUS] == AMD_OK_BUT_JUMBLED)
            {
               PRINTF (("OK, but jumbled\n")) ;
            }
            else
            {
               PRINTF (("unknown\n")) ;
            }

            /* statistics about the input matrix */
            PRI ("    n, dimension of A:                                  %.20g\n", n);
            PRI ("    nz, number of nonzeros in A:                        %.20g\n", ABIPInfo [AMD_NZ]) ;
            PRI ("    symmetry of A:                                      %.4f\n", ABIPInfo [AMD_SYMMETRY]) ;
            PRI ("    number of nonzeros on diagonal:                     %.20g\n", ABIPInfo [AMD_NZDIAG]) ;
            PRI ("    nonzeros in pattern of A+A' (excl. diagonal):       %.20g\n", ABIPInfo [AMD_NZ_A_PLUS_AT]) ;
            PRI ("    # dense rows/columns of A+A':                       %.20g\n", ABIPInfo [AMD_NDENSE]) ;

            /* statistics about AMD's behavior  */
```

```
            PRI ("    memory used, in bytes:                            %.20g\n", ABIPInfo
    [AMD_MEMORY]) ;
            PRI ("    # of memory compactions:                          %.20g\n", ABIPInfo
    [AMD_NCMPA]) ;

            /* statistics about the ordering quality */
            PRINTF (("\n"
           "    The following approximate statistics are for a subsequent\n"
           "    factorization of A(P,P) + A(P,P)'.  They are slight upper\n"
           "    bounds if there are no dense rows/columns in A+A', and become\n"
           "    looser if dense rows/columns exist.\n\n")) ;

            PRI ("    nonzeros in L (excluding diagonal):               %.20g\n", lnz) ;
            PRI ("    nonzeros in L (including diagonal):               %.20g\n", lnzd) ;
            PRI ("    # divide operations for LDL' or LU:               %.20g\n", ndiv) ;
            PRI ("    # multiply-subtract operations for LDL':          %.20g\n", nmultsubs_ldl) ;
            PRI ("    # multiply-subtract operations for LU:            %.20g\n", nmultsubs_lu) ;
            PRI ("    max nz. in any column of L (incl. diagonal):      %.20g\n", ABIPInfo
    [AMD_DMAX]) ;

            /* total flop counts for various factorizations */

            if (n >= 0 && ndiv >= 0 && nmultsubs_ldl >= 0 && nmultsubs_lu >= 0)
            {
               PRINTF (("\n"
                         "    chol flop count for real A, sqrt counted as 1 flop: %.20g\n"
                         "    LDL' flop count for real A:                         %.20g\n"
                         "    LDL' flop count for complex A:                      %.20g\n"
                         "    LU flop count for real A (with no pivoting):        %.20g\n"
                         "    LU flop count for complex A (with no pivoting):     %.20g\n\n",
                         n + ndiv + 2*nmultsubs_ldl,
                         ndiv + 2*nmultsubs_ldl,
                         9*ndiv + 8*nmultsubs_ldl,
                         ndiv + 2*nmultsubs_lu,
                         9*ndiv + 8*nmultsubs_lu)) ;
            }
 }
```

5.23 external/amd/amd_internal.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <limits.h>
#include <math.h>
#include "amd.h"
```

Macros

- #define EMPTY (-1)
- #define FLIP(i) (-(i)-2)
- #define UNFLIP(i) ((i < EMPTY) ? FLIP (i) : (i))
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
- #define IMPLIES(p, q) (!(p) || (q))
- #define TRUE (1)
- #define FALSE (0)
- #define PRIVATE static
- #define GLOBAL
- #define EMPTY (-1)
- #define ABIP_NULL 0
- #define SIZE_T_MAX ((size_t) (-1))
- #define Int int
- #define ID "%d"
- #define Int_MAX INT_MAX

- #define AMD_order amd_order
- #define AMD_defaults amd_defaults
- #define AMD_control amd_control
- #define AMD_info amd_info
- #define AMD_1 amd_1
- #define AMD_2 amd_2
- #define AMD_valid amd_valid
- #define AMD_aat amd_aat
- #define AMD_postorder amd_postorder
- #define AMD_post_tree amd_post_tree
- #define AMD_dump amd_dump
- #define AMD_debug amd_debug
- #define AMD_debug_init amd_debug_init
- #define AMD_preprocess amd_preprocess
- #define PRINTF(params) { if (amd_printf != ABIP_NULL) (void) amd_printf params ; }
- #define ASSERT(expression)
- #define AMD_DEBUG0(params)
- #define AMD_DEBUG1(params)
- #define AMD_DEBUG2(params)
- #define AMD_DEBUG3(params)
- #define AMD_DEBUG4(params)

Functions

- GLOBAL size_t AMD_aat (Int n, const Int Ap[ ], const Int Ai[ ], Int Len[ ], Int Tp[ ], abip_float ABIPInfo[ ])
- GLOBAL void AMD_1 (Int n, const Int Ap[ ], const Int Ai[ ], Int P[ ], Int Pinv[ ], Int Len[ ], Int slen, Int S[ ], abip_float Control[ ], abip_float ABIPInfo[ ])
- GLOBAL void AMD_postorder (Int nn, Int Parent[ ], Int Npiv[ ], Int Fsize[ ], Int Order[ ], Int Child[ ], Int Sibling[ ], Int Stack[ ])
- GLOBAL Int AMD_post_tree (Int root, Int k, Int Child[ ], const Int Sibling[ ], Int Order[ ], Int Stack[ ])
- GLOBAL void AMD_preprocess (Int n, const Int Ap[ ], const Int Ai[ ], Int Rp[ ], Int Ri[ ], Int W[ ], Int Flag[ ])

5.23.1 Macro Definition Documentation

5.23.1.1 ABIP_NULL

```
#define ABIP_NULL 0
```

Definition at line 138 of file amd_internal.h.

5.23.1.2 AMD_1

```
#define AMD_1 amd_1
```

Definition at line 187 of file amd_internal.h.

5.23.1.3 AMD_2

```
#define AMD_2 amd_2
```

Definition at line 188 of file amd_internal.h.

5.23.1.4 AMD_aat

```
#define AMD_aat amd_aat
```

Definition at line 190 of file amd_internal.h.

5.23.1.5 AMD_control

```
#define AMD_control amd_control
```

Definition at line 185 of file amd_internal.h.

5.23.1.6 AMD_debug

```
#define AMD_debug amd_debug
```

Definition at line 194 of file amd_internal.h.

5.23.1.7 AMD_DEBUG0

```
#define AMD_DEBUG0(
              params )
```

Definition at line 328 of file amd_internal.h.

5.23.1.8 AMD_DEBUG1

```
#define AMD_DEBUG1(
              params )
```

Definition at line 329 of file amd_internal.h.

5.23.1.9 AMD_DEBUG2

```
#define AMD_DEBUG2(
              params )
```

Definition at line 330 of file amd_internal.h.

5.23.1.10 AMD_DEBUG3

```
#define AMD_DEBUG3(
              params )
```

Definition at line 331 of file amd_internal.h.

5.23.1.11 AMD_DEBUG4

```
#define AMD_DEBUG4(
              params )
```

Definition at line 332 of file amd_internal.h.

5.23.1.12 AMD_debug_init

```
#define AMD_debug_init amd_debug_init
```

Definition at line 195 of file amd_internal.h.

5.23.1.13 AMD_defaults

```
#define AMD_defaults amd_defaults
```

Definition at line 184 of file amd_internal.h.

5.23.1.14 AMD_dump

```
#define AMD_dump amd_dump
```

Definition at line 193 of file amd_internal.h.

5.23.1.15 AMD_info

```
#define AMD_info amd_info
```

Definition at line 186 of file amd_internal.h.

5.23.1.16 AMD_order

```
#define AMD_order amd_order
```

Definition at line 183 of file amd_internal.h.

5.23.1.17 AMD_post_tree

```
#define AMD_post_tree amd_post_tree
```

Definition at line 192 of file amd_internal.h.

5.23.1.18 AMD_postorder

```
#define AMD_postorder amd_postorder
```

Definition at line 191 of file amd_internal.h.

5.23.1.19 AMD_preprocess

```
#define AMD_preprocess amd_preprocess
```

Definition at line 196 of file amd_internal.h.

5.23.1.20 AMD_valid

```
#define AMD_valid amd_valid
```

Definition at line 189 of file amd_internal.h.

5.23.1.21 ASSERT

```
#define ASSERT(
            expression )
```

Definition at line 327 of file amd_internal.h.

5.23.1.22 EMPTY [1/2]

```
#define EMPTY (-1)
```

Definition at line 129 of file amd_internal.h.

5.23.1.23 EMPTY [2/2]

```
#define EMPTY (-1)
```

Definition at line 129 of file amd_internal.h.

5.23.1.24 FALSE

```
#define FALSE (0)
```

Definition at line 126 of file amd_internal.h.

5.23.1.25 FLIP

```
#define FLIP(
            i ) (-(i)-2)
```

Definition at line 106 of file amd_internal.h.

5.23.1.26 GLOBAL

```
#define GLOBAL
```

Definition at line 128 of file amd_internal.h.

5.23.1.27 ID

```
#define ID "%d"
```

Definition at line 180 of file amd_internal.h.

5.23.1.28 IMPLIES

```
#define IMPLIES(
              p,
              q ) (!(p) || (q))
```

Definition at line 114 of file amd_internal.h.

5.23.1.29 Int

```
#define Int int
```

Definition at line 179 of file amd_internal.h.

5.23.1.30 Int_MAX

```
#define Int_MAX INT_MAX
```

Definition at line 181 of file amd_internal.h.

5.23.1.31 MAX

```
#define MAX(
              a,
              b ) (((a) > (b)) ? (a) : (b))
```

Definition at line 110 of file amd_internal.h.

5.23.1.32 MIN

```
#define MIN(
              a,
              b ) (((a) < (b)) ? (a) : (b))
```

Definition at line 111 of file amd_internal.h.

5.23.1.33 PRINTF

```
#define PRINTF(
              params ) { if (amd_printf != ABIP_NULL) (void) amd_printf params ; }
```

Definition at line 205 of file amd_internal.h.

5.23.1.34 PRIVATE

```
#define PRIVATE static
```

Definition at line 127 of file amd_internal.h.

5.23.1.35 SIZE_T_MAX

```
#define SIZE_T_MAX ((size_t) (-1))
```

Definition at line 146 of file amd_internal.h.

5.23.1.36 TRUE

```
#define TRUE (1)
```

Definition at line 125 of file amd_internal.h.

5.23.1.37 UNFLIP

```
#define UNFLIP(
              i ) ((i < EMPTY) ? FLIP (i) : (i))
```

Definition at line 107 of file amd_internal.h.

5.23.2 Function Documentation

5.23.2.1 AMD_1()

```
GLOBAL void AMD_1 (
            Int n,
            const Int Ap[ ],
            const Int Ai[ ],
            Int P[ ],
            Int Pinv[ ],
            Int Len[ ],
            Int slen,
            Int S[ ],
            abip_float Control[ ],
            abip_float ABIPInfo[ ] )
```

Definition at line 29 of file amd_1.c.

5.23.2.2 AMD_aat()

```
GLOBAL size_t AMD_aat (
            Int n,
            const Int Ap[ ],
            const Int Ai[ ],
            Int Len[ ],
            Int Tp[ ],
            abip_float ABIPInfo[ ] )
```

Definition at line 20 of file amd_aat.c.

5.23.2.3 AMD_post_tree()

```
GLOBAL Int AMD_post_tree (
            Int root,
            Int k,
            Int Child[ ],
            const Int Sibling[ ],
            Int Order[ ],
            Int Stack[ ] )
```

5.23.2.4 AMD_postorder()

```
GLOBAL void AMD_postorder (
            Int nn,
            Int Parent[ ],
            Int Npiv[ ],
            Int Fsize[ ],
            Int Order[ ],
            Int Child[ ],
            Int Sibling[ ],
            Int Stack[ ] )
```

Definition at line 15 of file amd_postorder.c.

5.23.2.5 AMD_preprocess()

```
GLOBAL void AMD_preprocess (
            Int n,
            const Int Ap[ ],
            const Int Ai[ ],
            Int Rp[ ],
            Int Ri[ ],
            Int W[ ],
            Int Flag[ ] )
```

Definition at line 29 of file amd_preprocess.c.

5.24 amd_internal.h

Go to the documentation of this file.

```
/* ========================================================================= */
/* === amd_internal.h ====================================================== */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                      */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* This file is for internal use in AMD itself, and does not normally need to
 * be included in user code (it is included in UMFPACK, however).   All others
 * should use amd.h instead.
 */

/* ========================================================================= */
/* === NDEBUG ============================================================== */
/* ========================================================================= */

/*
 * Turning on debugging takes some work (see below).   If you do not edit this
 * file, then debugging is always turned off, regardless of whether or not
 * -DNDEBUG is specified in your compiler options.
 *
 * If AMD is being compiled as a mexFunction, then MATLAB_MEX_FILE is defined,
 * and mxAssert is used instead of assert.  If debugging is not enabled, no
 * MATLAB include files or functions are used.  Thus, the AMD library libamd.a
 * can be safely used in either a stand-alone C program or in another
 * mexFunction, without any change.
 */

/*
    AMD will be exceedingly slow when running in debug mode.  The next three
    lines ensure that debugging is turned off.
*/
#ifndef NDEBUG
#define NDEBUG
#endif

/*
    To enable debugging, uncomment the following line:
#undef NDEBUG
*/

/* ------------------------------------------------------------------------- */
/* ANSI include files */
/* ------------------------------------------------------------------------- */

/* from stdlib.h:  size_t, malloc, free, realloc, and calloc */
#include <stdlib.h>

#if !defined(NPRINT) || !defined(NDEBUG)
/* from stdio.h:  printf.  Not included if NPRINT is defined at compile time.
 * fopen and fscanf are used when debugging. */
#include <stdio.h>
#endif

/* from limits.h:  INT_MAX and LONG_MAX */
#include <limits.h>
```

```
/* from math.h: sqrt */
#include <math.h>

/* ------------------------------------------------------------------------- */
/* MATLAB include files (only if being used in or via MATLAB) */
/* ------------------------------------------------------------------------- */

#ifdef MATLAB_MEX_FILE
#include "matrix.h"
#include "mex.h"
#endif

/* ------------------------------------------------------------------------- */
/* basic definitions */
/* ------------------------------------------------------------------------- */

#ifdef FLIP
#undef FLIP
#endif

#ifdef MAX
#undef MAX
#endif

#ifdef MIN
#undef MIN
#endif

#ifdef EMPTY
#undef EMPTY
#endif

#ifdef GLOBAL
#undef GLOBAL
#endif

#ifdef PRIVATE
#undef PRIVATE
#endif

/* FLIP is a "negation about -1", and is used to mark an integer i that is
 * normally non-negative.  FLIP (EMPTY) is EMPTY.  FLIP of a number > EMPTY
 * is negative, and FLIP of a number < EMTPY is positive.  FLIP (FLIP (i)) = i
 * for all integers i.  UNFLIP (i) is >= EMPTY. */
#define EMPTY (-1)
#define FLIP(i) (-(i)-2)
#define UNFLIP(i) ((i < EMPTY) ? FLIP (i) : (i))

/* for integer MAX/MIN, or for doubles when we don't care how NaN's behave: */
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
#define MIN(a,b) (((a) < (b)) ? (a) : (b))

/* logical expression of p implies q: */
#define IMPLIES(p,q) (!(p) || (q))

/* Note that the IBM RS 6000 xlc predefines TRUE and FALSE in <types.h>. */
/* The Compaq Alpha also predefines TRUE and FALSE. */
#ifdef TRUE
#undef TRUE
#endif
#ifdef FALSE
#undef FALSE
#endif

#define TRUE (1)
#define FALSE (0)
#define PRIVATE static
#define GLOBAL
#define EMPTY (-1)

/* Note that Linux's gcc 2.96 defines NULL as ((void *) 0), but other */
/* compilers (even gcc 2.95.2 on Solaris) define NULL as 0 or (0).  We */
/* need to use the ANSI standard value of 0. */
#ifdef ABIP_NULL
#undef ABIP_NULL
#endif

#define ABIP_NULL 0

/* largest value of size_t */
#ifndef SIZE_T_MAX
#ifdef SIZE_MAX
/* C99 only */
#define SIZE_T_MAX SIZE_MAX
#else
#define SIZE_T_MAX ((size_t) (-1))
```

```
#endif
#endif

/* ------------------------------------------------------------------------- */
/* integer type for AMD: int or SuiteSparse_long */
/* ------------------------------------------------------------------------- */

#include "amd.h"

#if defined (DLONG) || defined (ZLONG)

#define Int SuiteSparse_long
#define ID  SuiteSparse_long_id
#define Int_MAX SuiteSparse_long_max

#define AMD_order amd_l_order
#define AMD_defaults amd_l_defaults
#define AMD_control amd_l_control
#define AMD_info amd_l_info
#define AMD_1 amd_l1
#define AMD_2 amd_l2
#define AMD_valid amd_l_valid
#define AMD_aat amd_l_aat
#define AMD_postorder amd_l_postorder
#define AMD_post_tree amd_l_post_tree
#define AMD_dump amd_l_dump
#define AMD_debug amd_l_debug
#define AMD_debug_init amd_l_debug_init
#define AMD_preprocess amd_l_preprocess

#else

#define Int int
#define ID "%d"
#define Int_MAX INT_MAX

#define AMD_order amd_order
#define AMD_defaults amd_defaults
#define AMD_control amd_control
#define AMD_info amd_info
#define AMD_1 amd_1
#define AMD_2 amd_2
#define AMD_valid amd_valid
#define AMD_aat amd_aat
#define AMD_postorder amd_postorder
#define AMD_post_tree amd_post_tree
#define AMD_dump amd_dump
#define AMD_debug amd_debug
#define AMD_debug_init amd_debug_init
#define AMD_preprocess amd_preprocess

#endif

/* ========================================================================= */
/* === PRINTF macro ======================================================== */
/* ========================================================================= */

/* All output goes through the PRINTF macro.  */
#define PRINTF(params) { if (amd_printf != ABIP_NULL) (void) amd_printf params ; }

/* ------------------------------------------------------------------------- */
/* AMD routine definitions (not user-callable) */
/* ------------------------------------------------------------------------- */

GLOBAL size_t AMD_aat
(
            Int n,
            const Int Ap [ ],
            const Int Ai [ ],
            Int Len [ ],
            Int Tp [ ],
            abip_float ABIPInfo [ ]
) ;

GLOBAL void AMD_1
(
            Int n,
            const Int Ap [ ],
            const Int Ai [ ],
            Int P [ ],
            Int Pinv [ ],
            Int Len [ ],
            Int slen,
            Int S [ ],
            abip_float Control [ ],
            abip_float ABIPInfo [ ]
) ;
```

```

GLOBAL void AMD_postorder
(
            Int nn,
            Int Parent [ ],
            Int Npiv [ ],
            Int Fsize [ ],
            Int Order [ ],
            Int Child [ ],
            Int Sibling [ ],
            Int Stack [ ]
) ;

GLOBAL Int AMD_post_tree
(
            Int root,
            Int k,
            Int Child [ ],
            const Int Sibling [ ],
            Int Order [ ],
            Int Stack [ ]

            #ifndef NDEBUG
            , Int nn
            #endif
) ;

GLOBAL void AMD_preprocess
(
            Int n,
            const Int Ap [ ],
            const Int Ai [ ],
            Int Rp [ ],
            Int Ri [ ],
            Int W [ ],
            Int Flag [ ]
) ;

/* ------------------------------------------------------------------------- */
/* debugging definitions */
/* ------------------------------------------------------------------------- */

#ifndef NDEBUG

/* from assert.h:  assert macro */
#include <assert.h>

#ifndef EXTERN
#define EXTERN extern
#endif

EXTERN Int AMD_debug ;

GLOBAL void AMD_debug_init ( char *s ) ;

GLOBAL void AMD_dump
(
            Int n,
            Int Pe [ ],
            Int Iw [ ],
            Int Len [ ],
            Int iwlen,
            Int pfree,
            Int Nv [ ],
            Int Next [ ],
            Int Last [ ],
            Int Head [ ],
            Int Elen [ ],
            Int Degree [ ],
            Int W [ ],
            Int nel
) ;

#ifdef ASSERT
#undef ASSERT
#endif

/* Use mxAssert if AMD is compiled into a mexFunction */
#ifdef MATLAB_MEX_FILE
#define ASSERT(expression) (mxAssert ((expression), ""))
#else
#define ASSERT(expression) (assert (expression))
#endif

#define AMD_DEBUG0(params) { PRINTF (params) ; }
#define AMD_DEBUG1(params) { if (AMD_debug >= 1) PRINTF (params) ; }
#define AMD_DEBUG2(params) { if (AMD_debug >= 2) PRINTF (params) ; }
```

```
#define AMD_DEBUG3(params) { if (AMD_debug >= 3) PRINTF (params) ; }
#define AMD_DEBUG4(params) { if (AMD_debug >= 4) PRINTF (params) ; }

#else

/* no debugging */
#define ASSERT(expression)
#define AMD_DEBUG0(params)
#define AMD_DEBUG1(params)
#define AMD_DEBUG2(params)
#define AMD_DEBUG3(params)
#define AMD_DEBUG4(params)

#endif
```

5.25 external/amd/amd_order.c File Reference

`#include "amd_internal.h"`

Functions

- GLOBAL Int AMD_order (Int n, const Int Ap[ ], const Int Ai[ ], Int P[ ], abip_float Control[ ], abip_float ABIPInfo[ ])

5.25.1 Function Documentation

5.25.1.1 AMD_order()

```
GLOBAL Int AMD_order (
        Int n,
        const Int Ap[ ],
        const Int Ai[ ],
        Int P[ ],
        abip_float Control[ ],
        abip_float ABIPInfo[ ] )
```

Definition at line 21 of file amd_order.c.

5.26 amd_order.c

Go to the documentation of this file.

```
00001 /* ========================================================================= */
00002 /* === AMD_order =========================================================== */
00003 /* ========================================================================= */
00004
00005 /* ------------------------------------------------------------------------- */
00006 /* AMD, Copyright (c) Timothy A. Davis,                                      */
00007 /* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
00008 /* email: DrTimothyAldenDavis@gmail.com                                      */
00009 /* ------------------------------------------------------------------------- */
00010
00011 /* User-callable AMD minimum degree ordering routine.  See amd.h for
00012  * documentation.
00013  */
```

```
#include "amd_internal.h"

/* ========================================================================= */
/* === AMD_order =========================================================== */
/* ========================================================================= */

GLOBAL Int AMD_order
(
            Int n,
            const Int Ap [ ],
            const Int Ai [ ],
            Int P [ ],
            abip_float Control [ ],
            abip_float ABIPInfo [ ]
)
{
            Int *Len;
            Int *S;
            Int nz;
            Int i;
            Int *Pinv;
            Int info;
            Int status;

            Int *Rp;
            Int *Ri;
            Int *Cp;
            Int *Ci;
            Int ok;

            size_t nzaat;
            size_t slen;

            abip_float mem = 0 ;

            #ifndef NDEBUG
            AMD_debug_init ("amd") ;
            #endif

            /* clear the ABIPInfo array, if it exists */
            info = ABIPInfo != (abip_float *) ABIP_NULL ;

            if (info)
            {
               for (i = 0 ; i < AMD_INFO ; i++)
               {
                           ABIPInfo [i] = EMPTY ;
               }
               ABIPInfo [AMD_N] = n ;
               ABIPInfo [AMD_STATUS] = AMD_OK ;
            }

            /* make sure inputs exist and n is >= 0 */
            if (Ai == (Int *) ABIP_NULL || Ap == (Int *) ABIP_NULL || P == (Int *) ABIP_NULL || n < 0)
            {
               if (info) ABIPInfo [AMD_STATUS] = AMD_INVALID ;
               return (AMD_INVALID) ;        /* arguments are invalid */
            }

            if (n == 0)
            {
               return (AMD_OK) ;      /* n is 0 so there's nothing to do */
            }

            nz = Ap [n] ;

            if (info)
            {
               ABIPInfo [AMD_NZ] = nz ;
            }

            if (nz < 0)
            {
               if (info) ABIPInfo [AMD_STATUS] = AMD_INVALID ;
               return (AMD_INVALID) ;
            }

            /* check if n or nz will cause size_t overflow */
            if (((size_t) n) >= SIZE_T_MAX / sizeof (Int) || ((size_t) nz) >= SIZE_T_MAX / sizeof
   (Int))
            {
               if (info) ABIPInfo [AMD_STATUS] = AMD_OUT_OF_MEMORY ;
               return (AMD_OUT_OF_MEMORY) ;        /* problem too large */
            }

            /* check the input matrix:  AMD_OK, AMD_INVALID, or AMD_OK_BUT_JUMBLED */
```

```
        status = AMD_valid (n, n, Ap, Ai) ;

        if (status == AMD_INVALID)
        {
           if (info) ABIPInfo [AMD_STATUS] = AMD_INVALID ;
           return (AMD_INVALID) ;        /* matrix is invalid */
        }

        /* allocate two size-n integer workspaces */
        Len  = amd_malloc (n * sizeof (Int)) ;
        Pinv = amd_malloc (n * sizeof (Int)) ;
        mem += n ;
        mem += n ;

        if (!Len || !Pinv)
        {
           /* :: out of memory :: */
           amd_free (Len) ;
           amd_free (Pinv) ;
           if (info) ABIPInfo [AMD_STATUS] = AMD_OUT_OF_MEMORY ;
           return (AMD_OUT_OF_MEMORY) ;
        }

        if (status == AMD_OK_BUT_JUMBLED)
        {
           /* sort the input matrix and remove duplicate entries */
           AMD_DEBUG1 (("Matrix is jumbled\n")) ;
           Rp = amd_malloc ((n+1) * sizeof (Int));
           Ri = amd_malloc (nz * sizeof (Int));
           mem += (n+1) ;
           mem += MAX (nz,1) ;

                    if (!Rp || !Ri)
           {
                      /* :: out of memory :: */
                      amd_free (Rp) ;
                      amd_free (Ri) ;
                      amd_free (Len) ;
                      amd_free (Pinv) ;

                               if (info) ABIPInfo [AMD_STATUS] = AMD_OUT_OF_MEMORY ;
                      return (AMD_OUT_OF_MEMORY) ;
           }

           /* use Len and Pinv as workspace to create R = A' */
           AMD_preprocess (n, Ap, Ai, Rp, Ri, Len, Pinv) ;
           Cp = Rp ;
           Ci = Ri ;
        }
        else
        {
           /* order the input matrix as-is.  No need to compute R = A' first */
           Rp = ABIP_NULL ;
           Ri = ABIP_NULL ;
           Cp = (Int *) Ap ;
           Ci = (Int *) Ai ;
        }

        /* --------------------------------------------------------------------- */
        /* determine the symmetry and count off-diagonal nonzeros in A+A' */
        /* --------------------------------------------------------------------- */

        nzaat = AMD_aat (n, Cp, Ci, Len, P, ABIPInfo) ;
        AMD_DEBUG1 (("nzaat: %g\n", (abip_float) nzaat)) ;
        ASSERT ((MAX (nz-n, 0) <= nzaat) && (nzaat <= 2 * (size_t) nz)) ;

        /* --------------------------------------------------------------------- */
        /* allocate workspace for matrix, elbow room, and 6 size-n vectors */
        /* --------------------------------------------------------------------- */

        S = ABIP_NULL ;
        slen = nzaat ;                      /* space for matrix */
        ok = ((slen + nzaat/5) >= slen) ;   /* check for size_t overflow */
        slen += nzaat/5 ;           /* add elbow room */

        for (i = 0 ; ok && i < 7 ; i++)
        {
           ok = ((slen + n) > slen) ;   /* check for size_t overflow */
           slen += n ;          /* size-n elbow room, 6 size-n work */
        }

        mem += slen ;
        ok = ok && (slen < SIZE_T_MAX / sizeof (Int)) ;     /* check for overflow */
        ok = ok && (slen < Int_MAX) ;                            /* S[i] for Int i must be OK */

        if (ok)
```

```
            {
               S = amd_malloc (slen * sizeof (Int)) ;
            }

            AMD_DEBUG1 (("slen %g\n", (abip_float) slen)) ;

            if (!S)
            {
               /* :: out of memory :: (or problem too large) */
               amd_free (Rp) ;
               amd_free (Ri) ;
               amd_free (Len) ;
               amd_free (Pinv) ;
               if (info) ABIPInfo [AMD_STATUS] = AMD_OUT_OF_MEMORY ;
               return (AMD_OUT_OF_MEMORY) ;
            }

            if (info)
            {
               /* memory usage, in bytes. */
               ABIPInfo [AMD_MEMORY] = mem * sizeof (Int) ;
            }

            /* --------------------------------------------------------------------- */
            /* order the matrix */
            /* --------------------------------------------------------------------- */

            AMD_1 (n, Cp, Ci, P, Pinv, Len, slen, S, Control, ABIPInfo) ;

            /* --------------------------------------------------------------------- */
            /* free the workspace */
            /* --------------------------------------------------------------------- */

            amd_free (Rp) ;
            amd_free (Ri) ;
            amd_free (Len) ;
            amd_free (Pinv) ;
            amd_free (S) ;

            if (info) ABIPInfo [AMD_STATUS] = status ;
            return (status) ;       /* successful ordering */
}
```

5.27 external/amd/amd_post_tree.c File Reference

`#include "amd_internal.h"`

Functions

- GLOBAL Int AMD_post_tree (Int root, Int k, Int Child[ ], const Int Sibling[ ], Int Order[ ], Int Stack[ ], Int nn)

5.27.1 Function Documentation

5.27.1.1 AMD_post_tree()

```
GLOBAL Int AMD_post_tree (
            Int root,
            Int k,
            Int Child[ ],
            const Int Sibling[ ],
            Int Order[ ],
            Int Stack[ ],
            Int nn )
```

Definition at line 15 of file amd_post_tree.c.

5.28 amd_post_tree.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_post_tree ======================================================= */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                    */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* Post-ordering of a supernodal elimination tree.  */

#include "amd_internal.h"

GLOBAL Int AMD_post_tree
(
        Int root,          /* root of the tree */
        Int k,             /* start numbering at k */
        Int Child [ ],         /* input argument of size nn, undefined on
                 * output.  Child [i] is the head of a link
                 * list of all nodes that are children of node
                 * i in the tree. */
        const Int Sibling [ ],    /* input argument of size nn, not modified.
                 * If f is a node in the link list of the
                 * children of node i, then Sibling [f] is the
                 * next child of node i.
                 */
        Int Order [ ],         /* output order, of size nn.  Order [i] = k
                 * if node i is the kth node of the reordered
                 * tree. */
        Int Stack [ ]          /* workspace of size nn */
    #ifndef NDEBUG
        , Int nn           /* nodes are in the range 0..nn-1. */
    #endif
)
{
        Int f;
        Int head;
        Int h;
        Int i;

    #if 0
            /* --------------------------------------------------------------------- */
            /* recursive version (Stack [ ] is not used): */
            /* --------------------------------------------------------------------- */

            /* this is simple, but can caouse stack overflow if nn is large */
            i = root ;
            for (f = Child [i] ; f != EMPTY ; f = Sibling [f])
            {
            k = AMD_post_tree (f, k, Child, Sibling, Order, Stack, nn) ;
            }
            Order [i] = k++ ;
            return (k) ;
    #endif

        /* --------------------------------------------------------------------- */
        /* non-recursive version, using an explicit stack */
        /* --------------------------------------------------------------------- */

        /* push root on the stack */
        head = 0 ;
        Stack [0] = root ;

        while (head >= 0)
        {
        /* get head of stack */
        ASSERT (head < nn) ;
        i = Stack [head] ;
        AMD_DEBUG1 (("head of stack "ID" \n", i)) ;
        ASSERT (i >= 0 && i < nn) ;

        if (Child [i] != EMPTY)
        {
                /* the children of i are not yet ordered */
                /* push each child onto the stack in reverse order */
                /* so that small ones at the head of the list get popped first */
                /* and the biggest one at the end of the list gets popped last */
                for (f = Child [i] ; f != EMPTY ; f = Sibling [f])
                {
                head++ ;
                ASSERT (head < nn) ;
```

```
                ASSERT (f >= 0 && f < nn) ;
                }

                h = head ;
                ASSERT (head < nn) ;

                for (f = Child [i] ; f != EMPTY ; f = Sibling [f])
                {
                ASSERT (h > 0) ;
                Stack [h--] = f ;
                AMD_DEBUG1 (("push "ID" on stack\n", f)) ;
                ASSERT (f >= 0 && f < nn) ;
                }

                ASSERT (Stack [h] == i) ;

                /* delete child list so that i gets ordered next time we see it */
                Child [i] = EMPTY ;
        }
        else
        {
                /* the children of i (if there were any) are already ordered */
                /* remove i from the stack and order it.  Front i is kth front */
                head-- ;
                AMD_DEBUG1 (("pop "ID" order "ID"\n", i, k)) ;
                Order [i] = k++ ;
                ASSERT (k <= nn) ;
        }

        #ifndef NDEBUG
        AMD_DEBUG1 (("\nStack:")) ;

        for (h = head ; h >= 0 ; h--)
        {
                Int j = Stack [h] ;
                AMD_DEBUG1 ((" "ID, j)) ;
                ASSERT (j >= 0 && j < nn) ;
        }

        AMD_DEBUG1 (("\n\n")) ;
        ASSERT (head < nn) ;
        #endif
    }
        return (k) ;
}
```

5.29 external/amd/amd_postorder.c File Reference

`#include "amd_internal.h"`

Functions

- GLOBAL void AMD_postorder (Int nn, Int Parent[ ], Int Nv[ ], Int Fsize[ ], Int Order[ ], Int Child[ ], Int Sibling[ ], Int Stack[ ])

5.29.1 Function Documentation

5.29.1.1 AMD_postorder()

```
GLOBAL void AMD_postorder (
            Int nn,
            Int Parent[ ],
            Int Nv[ ],
            Int Fsize[ ],
            Int Order[ ],
            Int Child[ ],
            Int Sibling[ ],
            Int Stack[ ] )
```

Definition at line 15 of file amd_postorder.c.

5.30 amd_postorder.c

Go to the documentation of this file.

```
00001 /* ========================================================================= */
00002 /* === AMD_postorder ======================================================= */
00003 /* ========================================================================= */
00004
00005 /* ------------------------------------------------------------------------- */
00006 /* AMD, Copyright (c) Timothy A. Davis,                                       */
00007 /* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
00008 /* email: DrTimothyAldenDavis@gmail.com                                      */
00009 /* ------------------------------------------------------------------------- */
00010
00011 /* Perform a postordering (via depth-first search) of an assembly tree. */
00012
00013 #include "amd_internal.h"
00014
00015 GLOBAL void AMD_postorder
00016 (
00017         /* inputs, not modified on output: */
00018         Int nn,          /* nodes are in the range 0..nn-1 */
00019         Int Parent [ ],       /* Parent [j] is the parent of j, or EMPTY if root */
00020         Int Nv [ ],      /* Nv [j] > 0 number of pivots represented by node j,
00021                 * or zero if j is not a node. */
00022         Int Fsize [ ],        /* Fsize [j]: size of node j */
00023
00024         /* output, not defined on input: */
00025         Int Order [ ],        /* output post-order */
00026
00027         /* workspaces of size nn: */
00028         Int Child [ ],
00029         Int Sibling [ ],
00030         Int Stack [ ]
00031 )
00032 {
00033         Int i;
00034         Int j;
00035         Int k;
00036
00037         Int parent;
00038         Int frsize;
00039         Int f;
00040         Int fprev;
00041         Int maxfrsize;
00042         Int bigfprev;
00043         Int bigf;
00044         Int fnext;
00045
00046         for (j = 0 ; j < nn ; j++)
00047         {
00048         Child [j] = EMPTY ;
00049         Sibling [j] = EMPTY ;
00050         }
00051
00052         /* --------------------------------------------------------------------- */
00053         /* place the children in link lists - bigger elements tend to be last */
00054         /* --------------------------------------------------------------------- */
00055
00056         for (j = nn-1 ; j >= 0 ; j--)
00057         {
```

```
        if (Nv [j] > 0)
        {
                /* this is an element */
                parent = Parent [j] ;
                if (parent != EMPTY)
                {
                /* place the element in link list of the children its parent */
                /* bigger elements will tend to be at the end of the list */
                Sibling [j] = Child [parent] ;
                Child [parent] = j ;
                }
        }
        }

    #ifndef NDEBUG

        Int nels;
        Int ff;
        Int nchild;
    AMD_DEBUG1 (("\n\n================================ AMD_postorder:\n"));
    nels = 0;

    for (j = 0 ; j < nn ; j++)
    {
            if (Nv [j] > 0)
            {
            AMD_DEBUG1 (( ""ID" :  nels "ID" npiv "ID" size "ID" parent "ID" maxfr "ID"\n", j, nels, Nv [j], Fsize [j], Parent [j], Fsize [j])) ;

            /* this is an element */
            /* dump the link list of children */
            nchild = 0 ;
            AMD_DEBUG1 (("    Children: ")) ;

            for (ff = Child [j] ; ff != EMPTY ; ff = Sibling [ff])
            {
                    AMD_DEBUG1 ((ID" ", ff)) ;
                    ASSERT (Parent [ff] == j) ;
                    nchild++ ;
                    ASSERT (nchild < nn) ;
            }

            AMD_DEBUG1 (("\n")) ;
            parent = Parent [j] ;

            if (parent != EMPTY)
            {
                    ASSERT (Nv [parent] > 0) ;
            }

            nels++ ;
            }
    }

        AMD_DEBUG1 (("\n\nGo through the children of each node, and put\n" "the biggest child last in each list:\n")) ;

    #endif

        /* --------------------------------------------------------------------- */
        /* place the largest child last in the list of children for each node */
        /* --------------------------------------------------------------------- */

        for (i = 0 ; i < nn ; i++)
        {
        if (Nv [i] > 0 && Child [i] != EMPTY)
        {

            #ifndef NDEBUG

                Int nchild ;
                AMD_DEBUG1 (("Before partial sort, element "ID"\n", i)) ;
                nchild = 0 ;

                for (f = Child [i] ; f != EMPTY ; f = Sibling [f])
                {
                ASSERT (f >= 0 && f < nn) ;
                AMD_DEBUG1 (("      f: "ID"  size: "ID"\n", f, Fsize [f])) ;
                nchild++ ;
                ASSERT (nchild <= nn) ;
                }

            #endif

                /* find the biggest element in the child list */
                fprev = EMPTY ;
                maxfrsize = EMPTY ;
```

```
                bigfprev = EMPTY ;
                bigf = EMPTY ;

                for (f = Child [i] ; f != EMPTY ; f = Sibling [f])
                {
                ASSERT (f >= 0 && f < nn) ;
                frsize = Fsize [f] ;

                if (frsize >= maxfrsize)
                {
                        /* this is the biggest seen so far */
                        maxfrsize = frsize ;
                        bigfprev = fprev ;
                        bigf = f ;
                }

                fprev = f ;
                }

                ASSERT (bigf != EMPTY) ;

                fnext = Sibling [bigf] ;

                AMD_DEBUG1 (("bigf "ID" maxfrsize "ID" bigfprev "ID" fnext "ID" fprev " ID"\n", bigf,
        maxfrsize, bigfprev, fnext, fprev)) ;

                if (fnext != EMPTY)
                {
                /* if fnext is EMPTY then bigf is already at the end of list */

                if (bigfprev == EMPTY)
                {
                        /* delete bigf from the element of the list */
                        Child [i] = fnext ;
                }
                else
                {
                        /* delete bigf from the middle of the list */
                        Sibling [bigfprev] = fnext ;
                }

                /* put bigf at the end of the list */
                Sibling [bigf] = EMPTY ;
                ASSERT (Child [i] != EMPTY) ;
                ASSERT (fprev != bigf) ;
                ASSERT (fprev != EMPTY) ;
                Sibling [fprev] = bigf ;
                }

            #ifndef NDEBUG

                AMD_DEBUG1 (("After partial sort, element "ID"\n", i)) ;

                for (f = Child [i] ; f != EMPTY ; f = Sibling [f])
                {
                ASSERT (f >= 0 && f < nn) ;
                AMD_DEBUG1 (("        "ID"  "ID"\n", f, Fsize [f])) ;
                ASSERT (Nv [f] > 0) ;
                nchild-- ;
                }

                ASSERT (nchild == 0) ;

            #endif
        }
        }

        /* --------------------------------------------------------------------- */
        /* postorder the assembly tree */
        /* --------------------------------------------------------------------- */

        for (i = 0 ; i < nn ; i++)
        {
        Order [i] = EMPTY ;
        }

        k = 0 ;

        for (i = 0 ; i < nn ; i++)
        {
        if (Parent [i] == EMPTY && Nv [i] > 0)
        {
                AMD_DEBUG1 (("Root of assembly tree "ID"\n", i)) ;
                k = AMD_post_tree (i, k, Child, Sibling, Order, Stack
            #ifndef NDEBUG
            , nn
            #endif
```

```
            ) ;
        }
        }
}
```

5.31 external/amd/amd_preprocess.c File Reference

```
#include "amd_internal.h"
```

Functions

- GLOBAL void AMD_preprocess (Int n, const Int Ap[ ], const Int Ai[ ], Int Rp[ ], Int Ri[ ], Int W[ ], Int Flag[ ])

5.31.1 Function Documentation

5.31.1.1 AMD_preprocess()

```
GLOBAL void AMD_preprocess (
            Int n,
            const Int Ap[ ],
            const Int Ai[ ],
            Int Rp[ ],
            Int Ri[ ],
            Int W[ ],
            Int Flag[ ] )
```

Definition at line 29 of file amd_preprocess.c.

5.32 amd_preprocess.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_preprocess ====================================================== */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                      */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* Sorts, removes duplicate entries, and transposes from the nonzero pattern of
 * a column-form matrix A, to obtain the matrix R.  The input matrix can have
 * duplicate entries and/or unsorted columns (AMD_valid (n,Ap,Ai) must not be
 * AMD_INVALID).
 *
 * This input condition is NOT checked.  This routine is not user-callable.
 */

#include "amd_internal.h"

/* ========================================================================= */
/* === AMD_preprocess ====================================================== */
/* ========================================================================= */
```

```

/* AMD_preprocess does not check its input for errors or allocate workspace.
 * On input, the condition (AMD_valid (n,n,Ap,Ai) != AMD_INVALID) must hold.
 */

GLOBAL void AMD_preprocess
(
            Int n,        /* input matrix: A is n-by-n */
            const Int Ap [ ],    /* size n+1 */
            const Int Ai [ ],    /* size nz = Ap [n] */

            /* output matrix R: */
            Int Rp [ ],       /* size n+1 */
            Int Ri [ ],       /* size nz (or less, if duplicates present) */

            Int W [ ],        /* workspace of size n */
            Int Flag [ ] /* workspace of size n */
)
{

            /* --------------------------------------------------------------------- */
            /* local variables */
            /* --------------------------------------------------------------------- */

            Int i;
            Int j;
            Int p;
            Int p2;

            ASSERT (AMD_valid (n, n, Ap, Ai) != AMD_INVALID) ;

            /* --------------------------------------------------------------------- */
            /* count the entries in each row of A (excluding duplicates) */
            /* --------------------------------------------------------------------- */

            for (i = 0 ; i < n ; i++)
            {
               W [i] = 0 ;       /* # of nonzeros in row i (excl duplicates) */
               Flag [i] = EMPTY ;  /* Flag [i] = j if i appears in column j */
            }

            for (j = 0 ; j < n ; j++)
            {
               p2 = Ap [j+1] ;
               for (p = Ap [j] ; p < p2 ; p++)
               {
                           i = Ai [p] ;
                           if (Flag [i] != j)
                           {
                                  /* row index i has not yet appeared in column j */
                                  W [i]++ ;     /* one more entry in row i */
                                  Flag [i] = j ;        /* flag row index i as appearing in col j*/
                           }
               }
            }

            /* --------------------------------------------------------------------- */
            /* compute the row pointers for R */
            /* --------------------------------------------------------------------- */

            Rp [0] = 0 ;
            for (i = 0 ; i < n ; i++)
            {
               Rp [i+1] = Rp [i] + W [i] ;
            }

            for (i = 0 ; i < n ; i++)
            {
               W [i] = Rp [i] ;
               Flag [i] = EMPTY ;
            }

            /* --------------------------------------------------------------------- */
            /* construct the row form matrix R */
            /* --------------------------------------------------------------------- */

            /* R = row form of pattern of A */
            for (j = 0 ; j < n ; j++)
            {
               p2 = Ap [j+1] ;
               for (p = Ap [j] ; p < p2 ; p++)
               {
                           i = Ai [p] ;
                           if (Flag [i] != j)
                           {
                                  /* row index i has not yet appeared in column j */
                                  Ri [W [i]++] = j ;  /* put col j in row i */
```

```
                                Flag [i] = j ;        /* flag row index i as appearing in col j*/
                          }
                    }
                }

                #ifndef NDEBUG
                ASSERT (AMD_valid (n, n, Rp, Ri) == AMD_OK) ;

                for (j = 0 ; j < n ; j++)
                {
                    ASSERT (W [j] == Rp [j+1]) ;
                }

                #endif
}
```

5.33 external/amd/amd_valid.c File Reference

```
#include "amd_internal.h"
```

Functions

- GLOBAL Int AMD_valid (Int n_row, Int n_col, const Int Ap[ ], const Int Ai[ ])

5.33.1 Function Documentation

5.33.1.1 AMD_valid()

```
GLOBAL Int AMD_valid (
            Int n_row,
            Int n_col,
            const Int Ap[ ],
            const Int Ai[ ] )
```

Definition at line 38 of file amd_valid.c.

5.34 amd_valid.c

Go to the documentation of this file.

```
/* ========================================================================= */
/* === AMD_valid =========================================================== */
/* ========================================================================= */

/* ------------------------------------------------------------------------- */
/* AMD, Copyright (c) Timothy A. Davis,                                   */
/* Patrick R. Amestoy, and Iain S. Duff.  See ../README.txt for License.     */
/* email: DrTimothyAldenDavis@gmail.com                                      */
/* ------------------------------------------------------------------------- */

/* Check if a column-form matrix is valid or not.  The matrix A is
 * n_row-by-n_col.  The row indices of entries in column j are in
 * Ai [Ap [j] ... Ap [j+1]-1].  Required conditions are:
 *
 *  n_row >= 0
 *  n_col >= 0
```

```
 *  nz = Ap [n_col] >= 0        number of entries in the matrix
 *  Ap [0] == 0
 *  Ap [j] <= Ap [j+1] for all j in the range 0 to n_col.
 *      Ai [0 ... nz-1] must be in the range 0 to n_row-1.
 *
 * If any of the above conditions hold, AMD_INVALID is returned.  If the
 * following condition holds, AMD_OK_BUT_JUMBLED is returned (a warning,
 * not an error):
 *
 *  row indices in Ai [Ap [j] ... Ap [j+1]-1] are not sorted in ascending
 *      order, and/or duplicate entries exist.
 *
 * Otherwise, AMD_OK is returned.
 *
 * In v1.2 and earlier, this function returned TRUE if the matrix was valid
 * (now returns AMD_OK), or FALSE otherwise (now returns AMD_INVALID or
 * AMD_OK_BUT_JUMBLED).
 */

#include "amd_internal.h"

GLOBAL Int AMD_valid
(
        /* inputs, not modified on output: */
        Int n_row,       /* A is n_row-by-n_col */
        Int n_col,
        const Int Ap [ ],     /* column pointers of A, of size n_col+1 */
        const Int Ai [ ]      /* row indices of A, of size nz = Ap [n_col] */
)
{
        Int nz;
        Int j;
        Int p1;
        Int p2;
        Int ilast;
        Int i;
        Int p;
        Int result = AMD_OK ;

        if (n_row < 0 || n_col < 0 || Ap == ABIP_NULL || Ai == ABIP_NULL)
        {
        return (AMD_INVALID) ;
        }

        nz = Ap [n_col] ;
        if (Ap [0] != 0 || nz < 0)
        {
        /* column pointers must start at Ap [0] = 0, and Ap [n] must be >= 0 */
        AMD_DEBUG0 (("column 0 pointer bad or nz < 0\n")) ;
        return (AMD_INVALID) ;
        }

        for (j = 0 ; j < n_col ; j++)
        {
        p1 = Ap [j] ;
        p2 = Ap [j+1] ;
        AMD_DEBUG2 (("\nColumn: "ID" p1: "ID" p2: "ID"\n", j, p1, p2)) ;

        if (p1 > p2)
        {
                /* column pointers must be ascending */
                AMD_DEBUG0 (("column "ID" pointer bad\n", j)) ;
                return (AMD_INVALID) ;
        }

        ilast = EMPTY ;
        for (p = p1 ; p < p2 ; p++)
        {
                i = Ai [p] ;
                AMD_DEBUG3 (("row: "ID"\n", i)) ;

                if (i < 0 || i >= n_row)
                {
                /* row index out of range */
                AMD_DEBUG0 (("index out of range, col "ID" row "ID"\n", j, i));
                return (AMD_INVALID) ;
                }

                if (i <= ilast)
                {
                /* row index unsorted, or duplicate entry present */
                AMD_DEBUG1 (("index unsorted/dupl col "ID" row "ID"\n", j, i));
                result = AMD_OK_BUT_JUMBLED ;
                }

                ilast = i;
        }
```

```
00104         }
00105 
00106         return (result) ;
00107 }
```

5.35 external/ldl/ldl.c File Reference

```
#include "ldl.h"
```

Functions

- void LDL_symbolic (LDL_int n, LDL_int Ap[ ], LDL_int Ai[ ], LDL_int Lp[ ], LDL_int Parent[ ], LDL_int Lnz[ ], LDL_int Flag[ ], LDL_int P[ ], LDL_int Pinv[ ])
- LDL_int LDL_numeric (LDL_int n, LDL_int Ap[ ], LDL_int Ai[ ], abip_float Ax[ ], LDL_int Lp[ ], LDL_int Parent[ ], LDL_int Lnz[ ], LDL_int Li[ ], abip_float Lx[ ], abip_float D[ ], abip_float Y[ ], LDL_int Pattern[ ], LDL_int Flag[ ], LDL_int P[ ], LDL_int Pinv[ ])
- void LDL_lsolve (LDL_int n, abip_float X[ ], LDL_int Lp[ ], LDL_int Li[ ], abip_float Lx[ ])
- void LDL_dsolve (LDL_int n, abip_float X[ ], abip_float D[ ])
- void LDL_ltsolve (LDL_int n, abip_float X[ ], LDL_int Lp[ ], LDL_int Li[ ], abip_float Lx[ ])
- void LDL_perm (LDL_int n, abip_float X[ ], abip_float B[ ], LDL_int P[ ])
- void LDL_permt (LDL_int n, abip_float X[ ], abip_float B[ ], LDL_int P[ ])
- LDL_int LDL_valid_perm (LDL_int n, LDL_int P[ ], LDL_int Flag[ ])
- LDL_int LDL_valid_matrix (LDL_int n, LDL_int Ap[ ], LDL_int Ai[ ])

5.35.1 Function Documentation

5.35.1.1 LDL_dsolve()

```
void LDL_dsolve (
            LDL_int n,
            abip_float X[ ],
            abip_float D[ ] )
```

Definition at line 385 of file ldl.c.

5.35.1.2 LDL_lsolve()

```
void LDL_lsolve (
            LDL_int n,
            abip_float X[ ],
            LDL_int Lp[ ],
            LDL_int Li[ ],
            abip_float Lx[ ] )
```

Definition at line 357 of file ldl.c.

5.35.1.3 LDL_ltsolve()

```
void LDL_ltsolve (
            LDL_int n,
            abip_float X[ ],
            LDL_int Lp[ ],
            LDL_int Li[ ],
            abip_float Lx[ ] )
```

Definition at line 431 of file ldl.c.

5.35.1.4 LDL_numeric()

```
LDL_int LDL_numeric (
            LDL_int n,
            LDL_int Ap[ ],
            LDL_int Ai[ ],
            abip_float Ax[ ],
            LDL_int Lp[ ],
            LDL_int Parent[ ],
            LDL_int Lnz[ ],
            LDL_int Li[ ],
            abip_float Lx[ ],
            abip_float D[ ],
            abip_float Y[ ],
            LDL_int Pattern[ ],
            LDL_int Flag[ ],
            LDL_int P[ ],
            LDL_int Pinv[ ] )
```

Definition at line 262 of file ldl.c.

5.35.1.5 LDL_perm()

```
void LDL_perm (
            LDL_int n,
            abip_float X[ ],
            abip_float B[ ],
            LDL_int P[ ] )
```

Definition at line 459 of file ldl.c.

5.35.1.6 LDL_permt()

```
void LDL_permt (
            LDL_int n,
            abip_float X[ ],
            abip_float B[ ],
            LDL_int P[ ] )
```

Definition at line 506 of file ldl.c.

5.35.1.7 LDL_symbolic()

```
void LDL_symbolic (
            LDL_int n,
            LDL_int Ap[ ],
            LDL_int Ai[ ],
            LDL_int Lp[ ],
            LDL_int Parent[ ],
            LDL_int Lnz[ ],
            LDL_int Flag[ ],
            LDL_int P[ ],
            LDL_int Pinv[ ] )
```

Definition at line 187 of file ldl.c.

5.35.1.8 LDL_valid_matrix()

```
LDL_int LDL_valid_matrix (
            LDL_int n,
            LDL_int Ap[ ],
            LDL_int Ai[ ] )
```

Definition at line 603 of file ldl.c.

5.35.1.9 LDL_valid_perm()

```
LDL_int LDL_valid_perm (
            LDL_int n,
            LDL_int P[ ],
            LDL_int Flag[ ] )
```

Definition at line 553 of file ldl.c.

5.36 ldl.c

Go to the documentation of this file.

```
/* ========================================================================== */
/* === ldl.c: sparse LDL' factorization and solve package =================== */
/* ========================================================================== */

/* LDL:  a simple set of routines for sparse LDL' factorization.  These routines
 * are not terrifically fast (they do not use dense matrix kernels), but the
 * code is very short.  The purpose is to illustrate the algorithms in a very
 * concise manner, primarily for educational purposes.  Although the code is
 * very concise, this package is slightly faster than the built-in sparse
 * Cholesky factorization in MATLAB 7.0 (chol), when using the same input
 * permutation.
 *
 * The routines compute the LDL' factorization of a real sparse symmetric
 * matrix A (or PAP' if a permutation P is supplied), and solve upper
 * and lower triangular systems with the resulting L and D factors.  If A is
 * positive definite then the factorization will be accurate.  A can be
 * indefinite (with negative values on the diagonal D), but in this case no
 * guarantee of accuracy is provided, since no numeric pivoting is performed.
```

```
 *
 * The n-by-n sparse matrix A is in compressed-column form.  The nonzero values
 * in column j are stored in Ax [Ap [j] ... Ap [j+1]-1], with corresponding row
 * indices in Ai [Ap [j] ... Ap [j+1]-1].  Ap [0] = 0 is required, and thus
 * nz = Ap [n] is the number of nonzeros in A.  Ap is an int array of size n+1.
 * The int array Ai and the abip_float array Ax are of size nz.  This data structure
 * is identical to the one used by MATLAB, except for the following
 * generalizations.  The row indices in each column of A need not be in any
 * particular order, although they must be in the range 0 to n-1.  Duplicate
 * entries can be present; any duplicates are summed.  That is, if row index i
 * appears twice in a column j, then the value of A (i,j) is the sum of the two
 * entries.  The data structure used here for the input matrix A is more
 * flexible than MATLAB's, which requires sorted columns with no duplicate
 * entries.
 *
 * Only the diagonal and upper triangular part of A (or PAP' if a permutation
 * P is provided) is accessed.  The lower triangular parts of the matrix A or
 * PAP' can be present, but they are ignored.
 *
 * The optional input permutation is provided as an array P of length n.  If
 * P [k] = j, the row and column j of A is the kth row and column of PAP'.
 * If P is present then the factorization is LDL' = PAP' or L*D*L' = A(P,P) in
 * 0-based MATLAB notation.  If P is not present (a null pointer) then no
 * permutation is performed, and the factorization is LDL' = A.
 *
 * The lower triangular matrix L is stored in the same compressed-column
 * form (an int Lp array of size n+1, an int Li array of size Lp [n], and a
 * abip_float array Lx of the same size as Li).  It has a unit diagonal, which is
 * not stored.  The row indices in each column of L are always returned in
 * ascending order, with no duplicate entries.  This format is compatible with
 * MATLAB, except that it would be more convenient for MATLAB to include the
 * unit diagonal of L.  Doing so here would add additional complexity to the
 * code, and is thus omitted in the interest of keeping this code short and
 * readable.
 *
 * The elimination tree is held in the Parent [0..n-1] array.  It is normally
 * not required by the user, but it is required by ldl_numeric.  The diagonal
 * matrix D is held as an array D [0..n-1] of size n.
 *
 * --------------------
 * C-callable routines:
 * --------------------
 *
 *  ldl_symbolic:  Given the pattern of A, computes the Lp and Parent arrays
 *      required by ldl_numeric.  Takes time proportional to the number of
 *      nonzeros in L.  Computes the inverse Pinv of P if P is provided.
 *      Also returns Lnz, the count of nonzeros in each column of L below
 *      the diagonal (this is not required by ldl_numeric).
 *  ldl_numeric:  Given the pattern and numerical values of A, the Lp array,
 *      the Parent array, and P and Pinv if applicable, computes the
 *      pattern and numerical values of L and D.
 *  ldl_lsolve:  Solves Lx=b for a dense vector b.
 *  ldl_dsolve:  Solves Dx=b for a dense vector b.
 *  ldl_ltsolve: Solves L'x=b for a dense vector b.
 *  ldl_perm:    Computes x=Pb for a dense vector b.
 *  ldl_permt:   Computes x=P'b for a dense vector b.
 *  ldl_valid_perm:  checks the validity of a permutation vector
 *  ldl_valid_matrix:  checks the validity of the sparse matrix A
 *
 * ----------------------------
 * Limitations of this package:
 * ----------------------------
 *
 * In the interest of keeping this code simple and readable, ldl_symbolic and
 * ldl_numeric assume their inputs are valid.  You can check your own inputs
 * prior to calling these routines with the ldl_valid_perm and ldl_valid_matrix
 * routines.  Except for the two ldl_valid_* routines, no routine checks to see
 * if the array arguments are present (non-NULL).  Like all C routines, no
 * routine can determine if the arrays are long enough and don't overlap.
 *
 * The ldl_numeric does check the numerical factorization, however.  It returns
 * n if the factorization is successful.  If D (k,k) is zero, then k is
 * returned, and L is only partially computed.
 *
 * No pivoting to control fill-in is performed, which is often critical for
 * obtaining good performance.  I recommend that you compute the permutation P
 * using AMD or SYMAMD (approximate minimum degree ordering routines), or an
 * appropriate graph-partitioning based ordering.  See the ldldemo.m routine for
 * an example in MATLAB, and the ldlmain.c stand-alone C program for examples of
 * how to find P.  Routines for manipulating compressed-column matrices are
 * available in UMFPACK.  AMD, SYMAMD, UMFPACK, and this LDL package are all
 * available at http://www.suitesparse.com.
 *
 * -------------------------
 * Possible simplifications:
 * -------------------------
 *
```

```
 * These routines could be made even simpler with a few additional assumptions.
 * If no input permutation were performed, the caller would have to permute the
 * matrix first, but the computation of Pinv, and the use of P and Pinv could be
 * removed.  If only the diagonal and upper triangular part of A or PAP' are
 * present, then the tests in the "if (i < k)" statement in ldl_symbolic and
 * "if (i <= k)" in ldl_numeric, are always true, and could be removed (i can
 * equal k in ldl_symbolic, but then the body of the if statement would
 * correctly do no work since Flag [k] == k).  If we could assume that no
 * duplicate entries are present, then the statement Y [i] += Ax [p] could be
 * replaced with Y [i] = Ax [p] in ldl_numeric.
 *
 * --------------------------
 * Description of the method:
 * --------------------------
 *
 * LDL computes the symbolic factorization by finding the pattern of L one row
 * at a time.  It does this based on the following theory.  Consider a sparse
 * system Lx=b, where L, x, and b, are all sparse, and where L comes from a
 * Cholesky (or LDL') factorization.  The elimination tree (etree) of L is
 * defined as follows.  The parent of node j is the smallest k > j such that
 * L (k,j) is nonzero.  Node j has no parent if column j of L is completely zero
 * below the diagonal (j is a root of the etree in this case).  The nonzero
 * pattern of x is the union of the paths from each node i to the root, for
 * each nonzero b (i).  To compute the numerical solution to Lx=b, we can
 * traverse the columns of L corresponding to nonzero values of x.  This
 * traversal does not need to be done in the order 0 to n-1.  It can be done in
 * any "topological" order, such that x (i) is computed before x (j) if i is a
 * descendant of j in the elimination tree.
 *
 * The row-form of the LDL' factorization is shown in the MATLAB function
 * ldlrow.m in this LDL package.  Note that row k of L is found via a sparse
 * triangular solve of L (1:k-1, 1:k-1) \ A (1:k-1, k), to use 1-based MATLAB
 * notation.  Thus, we can start with the nonzero pattern of the kth column of
 * A (above the diagonal), follow the paths up to the root of the etree of the
 * (k-1)-by-(k-1) leading submatrix of L, and obtain the pattern of the kth row
 * of L.  Note that we only need the leading (k-1)-by-(k-1) submatrix of L to
 * do this.  The elimination tree can be constructed as we go.
 *
 * The symbolic factorization does the same thing, except that it discards the
 * pattern of L as it is computed.  It simply counts the number of nonzeros in
 * each column of L and then constructs the Lp index array when it's done.  The
 * symbolic factorization does not need to do this in topological order.
 * Compare ldl_symbolic with the first part of ldl_numeric, and note that the
 * while (len > 0) loop is not present in ldl_symbolic.
 *
 * Copyright (c) 2006 by Timothy A Davis, http://www.suitesparse.com.
 * All Rights Reserved.  Developed while on sabbatical
 * at Stanford University and Lawrence Berkeley National Laboratory.  Refer to
 * the README file for the License.
 */

#include "ldl.h"

/* ========================================================================== */
/* === ldl_symbolic ========================================================= */
/* ========================================================================== */

/* The input to this routine is a sparse matrix A, stored in column form, and
 * an optional permutation P.  The output is the elimination tree
 * and the number of nonzeros in each column of L.  Parent [i] = k if k is the
 * parent of i in the tree.  The Parent array is required by ldl_numeric.
 * Lnz [k] gives the number of nonzeros in the kth column of L, excluding the
 * diagonal.
 *
 * One workspace vector (Flag) of size n is required.
 *
 * If P is NULL, then it is ignored.  The factorization will be LDL' = A.
 * Pinv is not computed.  In this case, neither P nor Pinv are required by
 * ldl_numeric.
 *
 * If P is not NULL, then it is assumed to be a valid permutation.  If
 * row and column j of A is the kth pivot, the P [k] = j.  The factorization
 * will be LDL' = PAP', or A (p,p) in MATLAB notation.  The inverse permutation
 * Pinv is computed, where Pinv [j] = k if P [k] = j.  In this case, both P
 * and Pinv are required as inputs to ldl_numeric.
 *
 * The floating-point operation count of the subsequent call to ldl_numeric
 * is not returned, but could be computed after ldl_symbolic is done.  It is
 * the sum of (Lnz [k]) * (Lnz [k] + 2) for k = 0 to n-1.
 */

void LDL_symbolic
(
        LDL_int n,       /* A and L are n-by-n, where n >= 0 */
        LDL_int Ap [ ],      /* input of size n+1, not modified */
        LDL_int Ai [ ],      /* input of size nz=Ap[n], not modified */
        LDL_int Lp [ ],      /* output of size n+1, not defined on input */
```

```
        LDL_int Parent [ ],  /* output of size n, not defined on input */
        LDL_int Lnz [ ],     /* output of size n, not defined on input */
        LDL_int Flag [ ],    /* workspace of size n, not defn. on input or output */
        LDL_int P [ ],       /* optional input of size n */
        LDL_int Pinv [ ]     /* optional output of size n (used if P is not NULL) */
)
{
        LDL_int i;
        LDL_int k;
        LDL_int p;
        LDL_int kk;
        LDL_int p2 ;

        if (P)
        {
        /* If P is present then compute Pinv, the inverse of P */
        for (k = 0 ; k < n ; k++)
        {
                Pinv [P [k]] = k ;
        }
        }

        for (k = 0 ; k < n ; k++)
        {
        /* L(k,:) pattern: all nodes reachable in etree from nz in A(0:k-1,k) */
        Parent [k] = -1 ;           /* parent of k is not yet known */
        Flag [k] = k ;              /* mark node k as visited */
        Lnz [k] = 0 ;               /* count of nonzeros in column k of L */
        kk = (P) ? (P [k]) : (k) ;      /* kth original, or permuted, column */
        p2 = Ap [kk+1] ;

        for (p = Ap [kk] ; p < p2 ; p++)
        {
                /* A (i,k) is nonzero (original or permuted A) */
                i = (Pinv) ? (Pinv [Ai [p]]) : (Ai [p]) ;

                if (i < k)
                {
                /* follow path from i to root of etree, stop at flagged node */
                for ( ; Flag [i] != k ; i = Parent [i])
                {
                        /* find parent of i if not yet determined */
                        if (Parent [i] == -1) Parent [i] = k ;
                        Lnz [i]++ ;         /* L (k,i) is nonzero */
                        Flag [i] = k ;          /* mark i as visited */
                }
                }
        }
        }

        /* construct Lp index array from Lnz column counts */
        Lp [0] = 0 ;
        for (k = 0 ; k < n ; k++)
        {
        Lp [k+1] = Lp [k] + Lnz [k] ;
        }
}


/* ========================================================================== */
/* === ldl_numeric ========================================================== */
/* ========================================================================== */

/* Given a sparse matrix A (the arguments n, Ap, Ai, and Ax) and its symbolic
 * analysis (Lp and Parent, and optionally P and Pinv), compute the numeric LDL'
 * factorization of A or PAP'.  The outputs of this routine are arguments Li,
 * Lx, and D.  It also requires three size-n workspaces (Y, Pattern, and Flag).
 */

LDL_int LDL_numeric   /* returns n if successful, k if D (k,k) is zero */
(
        LDL_int n,       /* A and L are n-by-n, where n >= 0 */
        LDL_int Ap [ ],      /* input of size n+1, not modified */
        LDL_int Ai [ ],      /* input of size nz=Ap[n], not modified */
        abip_float Ax [ ],  /* input of size nz=Ap[n], not modified */
        LDL_int Lp [ ],      /* input of size n+1, not modified */
        LDL_int Parent [ ],  /* input of size n, not modified */
        LDL_int Lnz [ ],     /* output of size n, not defn. on input */
        LDL_int Li [ ],      /* output of size lnz=Lp[n], not defined on input */
        abip_float Lx [ ],    /* output of size lnz=Lp[n], not defined on input */
        abip_float D [ ],     /* output of size n, not defined on input */
        abip_float Y [ ],     /* workspace of size n, not defn. on input or output */
        LDL_int Pattern [ ], /* workspace of size n, not defn. on input or output */
        LDL_int Flag [ ],    /* workspace of size n, not defn. on input or output */
        LDL_int P [ ],       /* optional input of size n */
        LDL_int Pinv [ ]     /* optional input of size n */
)
```

```
{
        abip_float yi;
        abip_float l_ki ;

        LDL_int i;
        LDL_int k;
        LDL_int p;
        LDL_int kk;
        LDL_int p2;
        LDL_int len;
        LDL_int top;

        for (k = 0 ; k < n ; k++)
        {

        if(abip_is_interrupted())
        {
                abip_printf("Interrupt detected in factorization! \n");
                return -1;
            }

        /* compute nonzero Pattern of kth row of L, in topological order */
        Y [k] = 0.0 ;           /* Y(0:k) is now all zero */
        top = n ;           /* stack for pattern is empty */
        Flag [k] = k ;          /* mark node k as visited */
        Lnz [k] = 0 ;           /* count of nonzeros in column k of L */
        kk = (P) ? (P [k]) : (k) ;      /* kth original, or permuted, column */
        p2 = Ap [kk+1] ;

        for (p = Ap [kk] ; p < p2 ; p++)
        {
                i = (Pinv) ? (Pinv [Ai [p]]) : (Ai [p]) ;   /* get A(i,k) */

                if (i <= k)
                {
                Y [i] += Ax [p] ;  /* scatter A(i,k) into Y (sum duplicates) */
                for (len = 0 ; Flag [i] != k ; i = Parent [i])
                {
                        Pattern [len++] = i ;           /* L(k,i) is nonzero */
                        Flag [i] = k ;              /* mark i as visited */
                }
                while (len > 0) Pattern [--top] = Pattern [--len] ;
                }
        }

        /* compute numerical values kth row of L (a sparse triangular solve) */
        D [k] = Y [k] ;         /* get D(k,k) and clear Y(k) */
        Y [k] = 0.0 ;
        for ( ; top < n ; top++)
        {
                i = Pattern [top] ;         /* Pattern [top:n-1] is pattern of L(:,k) */
                yi = Y [i] ;                /* get and clear Y(i) */
                Y [i] = 0.0 ;
                p2 = Lp [i] + Lnz [i] ;

                for (p = Lp [i] ; p < p2 ; p++)
                {
                Y [Li [p]] -= Lx [p] * yi ;
                }

                l_ki = yi / D [i] ;         /* the nonzero entry L(k,i) */
                D [k] -= l_ki * yi ;
                Li [p] = k ;                /* store L(k,i) in column form of L */
                Lx [p] = l_ki ;
                Lnz [i]++ ;             /* increment count of nonzeros in col i */
        }

        if (D [k] == 0.0) return (k) ;          /* failure, D(k,k) is zero */
        }
        return (n) ;                    /* success, diagonal of D is all nonzero */
}


/* ========================================================================== */
/* === ldl_lsolve:  solve Lx=b ============================================== */
/* ========================================================================== */

void LDL_lsolve
(
        LDL_int n,       /* L is n-by-n, where n >= 0 */
        abip_float X [ ],     /* size n.  right-hand-side on input, soln. on output */
        LDL_int Lp [ ],      /* input of size n+1, not modified */
        LDL_int Li [ ],      /* input of size lnz=Lp[n], not modified */
        abip_float Lx [ ]     /* input of size lnz=Lp[n], not modified */
)
{
        LDL_int j;
```

```
        LDL_int p;
        LDL_int p2 ;

        for (j = 0 ; j < n ; j++)
        {
        p2 = Lp [j+1] ;
        for (p = Lp [j] ; p < p2 ; p++)
        {
                X [Li [p]] -= Lx [p] * X [j] ;
        }
        }
}


/* ========================================================================== */
/* === ldl_dsolve:  solve Dx=b ============================================== */
/* ========================================================================== */

void LDL_dsolve
(
        LDL_int n,        /* D is n-by-n, where n >= 0 */
        abip_float X [ ],      /* size n.  right-hand-side on input, soln. on output */
        abip_float D [ ]       /* input of size n, not modified */
)
{
        LDL_int j ;
        for (j = 0 ; j < n - 7 ; ++j)
        {
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ; ++j;
        X [j] /= D [j] ;
        }

        if (j < n - 3)
        {
            X [j] /= D [j] ; ++j;
            X [j] /= D [j] ; ++j;
            X [j] /= D [j] ; ++j;
            X [j] /= D [j] ; ++j;
        }

        if (j < n - 1)
        {
            X [j] /= D [j] ; ++j;
            X [j] /= D [j] ; ++j;
        }

        if (j < n)
        {
            X [j] /= D [j] ;
        }

}


/* ========================================================================== */
/* === ldl_ltsolve: solve L'x=b  ============================================ */
/* ========================================================================== */

void LDL_ltsolve
(
        LDL_int n,        /* L is n-by-n, where n >= 0 */
        abip_float X [ ],      /* size n.  right-hand-side on input, soln. on output */
        LDL_int Lp [ ],       /* input of size n+1, not modified */
        LDL_int Li [ ],       /* input of size lnz=Lp[n], not modified */
        abip_float Lx [ ]      /* input of size lnz=Lp[n], not modified */
)
{
        LDL_int j;
        LDL_int p;
        LDL_int p2;

        for (j = n-1 ; j >= 0 ; j--)
        {
        p2 = Lp [j+1] ;
        for (p = Lp [j] ; p < p2 ; p++)
        {
                X [j] -= Lx [p] * X [Li [p]] ;
        }
        }
}

```

```c

/* ========================================================================== */
/* === ldl_perm: permute a vector, x=Pb ===================================== */
/* ========================================================================== */

void LDL_perm
(
        LDL_int n,       /* size of X, B, and P */
        abip_float X [ ],     /* output of size n. */
        abip_float B [ ],     /* input of size n. */
        LDL_int P [ ]        /* input permutation array of size n. */
)
{
        LDL_int j ;
        for (j = 0 ; j < n - 7; ++j)
        {
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ;
        }

        if (j < n - 3)
        {
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        }

        if (j < n - 1)
        {
        X [j] = B [P [j]] ; ++j;
        X [j] = B [P [j]] ; ++j;
        }

        if (j < n)
        {
        X [j] = B [P [j]] ;
        }

}


/* ========================================================================== */
/* === ldl_permt: permute a vector, x=P'b =================================== */
/* ========================================================================== */

void LDL_permt
(
        LDL_int n,       /* size of X, B, and P */
        abip_float X [ ],     /* output of size n. */
        abip_float B [ ],     /* input of size n. */
        LDL_int P [ ]        /* input permutation array of size n. */
)
{
        LDL_int j;
        for (j = 0 ; j < n - 7; ++j)
        {
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ;
        }

        if (j < n - 3)
        {
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        }

        if (j < n - 1)
        {
        X [P [j]] = B [j] ; ++j;
        X [P [j]] = B [j] ; ++j;
        }

```

```
        if (j < n)
        {
        X [P [j]] = B [j] ;
        }
}


/* ========================================================================== */
/* === ldl_valid_perm: check if a permutation vector is valid =============== */
/* ========================================================================== */

/* returns 1 if valid, 0 otherwise */
LDL_int LDL_valid_perm
(
        LDL_int n,
        LDL_int P [ ],              /* input of size n, a permutation of 0:n-1 */
        LDL_int Flag [ ]            /* workspace of size n */
)
{
        LDL_int j;
        LDL_int k;

        if (n < 0 || !Flag)
        {
        return (0) ;        /* n must be >= 0, and Flag must be present */
        }

        if (!P)
        {
        return (1) ;        /* If ABIP_NULL, P is assumed to be the identity perm. */
        }

        for (j = 0 ; j < n ; j++)
        {
        Flag [j] = 0 ;      /* clear the Flag array */
        }

        for (k = 0 ; k < n ; k++)
        {
        j = P [k] ;

        if (j < 0 || j >= n || Flag [j] != 0)
        {
                return (0) ;    /* P is not valid */
        }
        Flag [j] = 1 ;
        }
        return (1) ;        /* P is valid */
}


/* ========================================================================== */
/* === ldl_valid_matrix: check if a sparse matrix is valid ================== */
/* ========================================================================== */

/* This routine checks to see if a sparse matrix A is valid for input to
 * ldl_symbolic and ldl_numeric.  It returns 1 if the matrix is valid, 0
 * otherwise.  A is in sparse column form.  The numerical values in column j
 * are stored in Ax [Ap [j] ... Ap [j+1]-1], with row indices in
 * Ai [Ap [j] ... Ap [j+1]-1].  The Ax array is not checked.
 */

LDL_int LDL_valid_matrix
(
        LDL_int n,
        LDL_int Ap [ ],
        LDL_int Ai [ ]
)
{
        LDL_int j;
        LDL_int p;

        if (n < 0 || !Ap || !Ai || Ap [0] != 0)
        {
        return (0) ;        /* n must be >= 0, and Ap and Ai must be present */
        }

        for (j = 0 ; j < n ; j++)
        {
        if (Ap [j] > Ap [j+1])
        {
                return (0) ;    /* Ap must be monotonically nondecreasing */
        }
        }

        for (p = 0 ; p < Ap [n] ; p++)
        {
```

```
        if (Ai [p] < 0 || Ai [p] >= n)
        {
                return (0) ;     /* row indices must be in the range 0 to n-1 */
        }
        }

        return (1) ;        /* matrix is valid */
}
```

5.37 external/ldl/ldl.h File Reference

```
#include "SuiteSparse_config.h"
```

Macros

- #define LDL_int int
- #define LDL_ID "%d"
- #define LDL_symbolic ldl_symbolic
- #define LDL_numeric ldl_numeric
- #define LDL_lsolve ldl_lsolve
- #define LDL_dsolve ldl_dsolve
- #define LDL_ltsolve ldl_ltsolve
- #define LDL_perm ldl_perm
- #define LDL_permt ldl_permt
- #define LDL_valid_perm ldl_valid_perm
- #define LDL_valid_matrix ldl_valid_matrix
- #define LDL_DATE "May 4, 2016"
- #define LDL_VERSION_CODE(main, sub) ((main) ∗ 1000 + (sub))
- #define LDL_MAIN_VERSION 2
- #define LDL_SUB_VERSION 2
- #define LDL_SUBSUB_VERSION 6
- #define LDL_VERSION LDL_VERSION_CODE(LDL_MAIN_VERSION,LDL_SUB_VERSION)

Functions

- void ldl_symbolic (int n, int Ap[ ], int Ai[ ], int Lp[ ], int Parent[ ], int Lnz[ ], int Flag[ ], int P[ ], int Pinv[ ])
- int ldl_numeric (int n, int Ap[ ], int Ai[ ], abip_float Ax[ ], int Lp[ ], int Parent[ ], int Lnz[ ], int Li[ ], abip_float Lx[ ], abip_float D[ ], abip_float Y[ ], int Pattern[ ], int Flag[ ], int P[ ], int Pinv[ ])
- void ldl_lsolve (int n, abip_float X[ ], int Lp[ ], int Li[ ], abip_float Lx[ ])
- void ldl_dsolve (int n, abip_float X[ ], abip_float D[ ])
- void ldl_ltsolve (int n, abip_float X[ ], int Lp[ ], int Li[ ], abip_float Lx[ ])
- void ldl_perm (int n, abip_float X[ ], abip_float B[ ], int P[ ])
- void ldl_permt (int n, abip_float X[ ], abip_float B[ ], int P[ ])
- int ldl_valid_perm (int n, int P[ ], int Flag[ ])
- int ldl_valid_matrix (int n, int Ap[ ], int Ai[ ])
- void ldl_l_symbolic (SuiteSparse_long n, SuiteSparse_long Ap[ ], SuiteSparse_long Ai[ ], SuiteSparse_long Lp[ ], SuiteSparse_long Parent[ ], SuiteSparse_long Lnz[ ], SuiteSparse_long Flag[ ], SuiteSparse_long P[ ], SuiteSparse_long Pinv[ ])
- SuiteSparse_long ldl_l_numeric (SuiteSparse_long n, SuiteSparse_long Ap[ ], SuiteSparse_long Ai[ ], abip_float Ax[ ], SuiteSparse_long Lp[ ], SuiteSparse_long Parent[ ], SuiteSparse_long Lnz[ ], SuiteSparse_long Li[ ], abip_float Lx[ ], abip_float D[ ], abip_float Y[ ], SuiteSparse_long Pattern[ ], SuiteSparse_long Flag[ ], SuiteSparse_long P[ ], SuiteSparse_long Pinv[ ])

- void ldl_l_lsolve (SuiteSparse_long n, abip_float X[ ], SuiteSparse_long Lp[ ], SuiteSparse_long Li[ ], abip_float Lx[ ])
- void ldl_l_dsolve (SuiteSparse_long n, abip_float X[ ], abip_float D[ ])
- void ldl_l_ltsolve (SuiteSparse_long n, abip_float X[ ], SuiteSparse_long Lp[ ], SuiteSparse_long Li[ ], abip_float Lx[ ])
- void ldl_l_perm (SuiteSparse_long n, abip_float X[ ], abip_float B[ ], SuiteSparse_long P[ ])
- void ldl_l_permt (SuiteSparse_long n, abip_float X[ ], abip_float B[ ], SuiteSparse_long P[ ])
- SuiteSparse_long ldl_l_valid_perm (SuiteSparse_long n, SuiteSparse_long P[ ], SuiteSparse_long Flag[ ])
- SuiteSparse_long ldl_l_valid_matrix (SuiteSparse_long n, SuiteSparse_long Ap[ ], SuiteSparse_long Ai[ ])

5.37.1 Macro Definition Documentation

5.37.1.1 LDL_DATE

```
#define LDL_DATE "May 4, 2016"
```

Definition at line 106 of file ldl.h.

5.37.1.2 LDL_dsolve

```
#define LDL_dsolve ldl_dsolve
```

Definition at line 32 of file ldl.h.

5.37.1.3 LDL_ID

```
#define LDL_ID "%d"
```

Definition at line 27 of file ldl.h.

5.37.1.4 LDL_int

```
#define LDL_int int
```

Definition at line 26 of file ldl.h.

5.37.1.5 LDL_lsolve

```
#define LDL_lsolve ldl_lsolve
```

Definition at line 31 of file ldl.h.

5.37.1.6 LDL_ltsolve

```
#define LDL_ltsolve ldl_ltsolve
```

Definition at line 33 of file ldl.h.

5.37.1.7 LDL_MAIN_VERSION

```
#define LDL_MAIN_VERSION 2
```

Definition at line 108 of file ldl.h.

5.37.1.8 LDL_numeric

```
#define LDL_numeric ldl_numeric
```

Definition at line 30 of file ldl.h.

5.37.1.9 LDL_perm

```
#define LDL_perm ldl_perm
```

Definition at line 34 of file ldl.h.

5.37.1.10 LDL_permt

```
#define LDL_permt ldl_permt
```

Definition at line 35 of file ldl.h.

5.37.1.11 LDL_SUB_VERSION

```
#define LDL_SUB_VERSION 2
```

Definition at line 109 of file ldl.h.

5.37.1.12 LDL_SUBSUB_VERSION

```
#define LDL_SUBSUB_VERSION 6
```

Definition at line 110 of file ldl.h.

5.37.1.13 LDL_symbolic

```
#define LDL_symbolic ldl_symbolic
```

Definition at line 29 of file ldl.h.

5.37.1.14 LDL_valid_matrix

```
#define LDL_valid_matrix ldl_valid_matrix
```

Definition at line 37 of file ldl.h.

5.37.1.15 LDL_valid_perm

```
#define LDL_valid_perm ldl_valid_perm
```

Definition at line 36 of file ldl.h.

5.37.1.16 LDL_VERSION

```
#define LDL_VERSION LDL_VERSION_CODE(LDL_MAIN_VERSION,LDL_SUB_VERSION)
```

Definition at line 111 of file ldl.h.

5.37.1.17 LDL_VERSION_CODE

```
#define LDL_VERSION_CODE(
              main,
              sub ) ((main) * 1000 + (sub))
```

Definition at line 107 of file ldl.h.

5.37.2 Function Documentation

5.37.2.1 ldl_dsolve()

```
void ldl_dsolve (
             int n,
             abip_float X[ ],
             abip_float D[ ] )
```

5.37.2.2 ldl_l_dsolve()

```
void ldl_l_dsolve (
             SuiteSparse_long n,
             abip_float X[ ],
             abip_float D[ ] )
```

5.37.2.3 ldl_l_lsolve()

```
void ldl_l_lsolve (
             SuiteSparse_long n,
             abip_float X[ ],
             SuiteSparse_long Lp[ ],
             SuiteSparse_long Li[ ],
             abip_float Lx[ ] )
```

5.37.2.4 ldl_l_ltsolve()

```
void ldl_l_ltsolve (
             SuiteSparse_long n,
             abip_float X[ ],
             SuiteSparse_long Lp[ ],
             SuiteSparse_long Li[ ],
             abip_float Lx[ ] )
```

5.37.2.5 ldl_l_numeric()

```
SuiteSparse_long ldl_l_numeric (
            SuiteSparse_long n,
            SuiteSparse_long Ap[ ],
            SuiteSparse_long Ai[ ],
            abip_float Ax[ ],
            SuiteSparse_long Lp[ ],
            SuiteSparse_long Parent[ ],
            SuiteSparse_long Lnz[ ],
            SuiteSparse_long Li[ ],
            abip_float Lx[ ],
            abip_float D[ ],
            abip_float Y[ ],
            SuiteSparse_long Pattern[ ],
            SuiteSparse_long Flag[ ],
            SuiteSparse_long P[ ],
            SuiteSparse_long Pinv[ ] )
```

5.37.2.6 ldl_l_perm()

```
void ldl_l_perm (
            SuiteSparse_long n,
            abip_float X[ ],
            abip_float B[ ],
            SuiteSparse_long P[ ] )
```

5.37.2.7 ldl_l_permt()

```
void ldl_l_permt (
            SuiteSparse_long n,
            abip_float X[ ],
            abip_float B[ ],
            SuiteSparse_long P[ ] )
```

5.37.2.8 ldl_l_symbolic()

```
void ldl_l_symbolic (
            SuiteSparse_long n,
            SuiteSparse_long Ap[ ],
            SuiteSparse_long Ai[ ],
            SuiteSparse_long Lp[ ],
            SuiteSparse_long Parent[ ],
            SuiteSparse_long Lnz[ ],
            SuiteSparse_long Flag[ ],
            SuiteSparse_long P[ ],
            SuiteSparse_long Pinv[ ] )
```

5.37.2.9 ldl_l_valid_matrix()

```
SuiteSparse_long ldl_l_valid_matrix (
            SuiteSparse_long n,
            SuiteSparse_long Ap[ ],
            SuiteSparse_long Ai[ ] )
```

5.37.2.10 ldl_l_valid_perm()

```
SuiteSparse_long ldl_l_valid_perm (
            SuiteSparse_long n,
            SuiteSparse_long P[ ],
            SuiteSparse_long Flag[ ] )
```

5.37.2.11 ldl_lsolve()

```
void ldl_lsolve (
            int n,
            abip_float X[ ],
            int Lp[ ],
            int Li[ ],
            abip_float Lx[ ] )
```

5.37.2.12 ldl_ltsolve()

```
void ldl_ltsolve (
            int n,
            abip_float X[ ],
            int Lp[ ],
            int Li[ ],
            abip_float Lx[ ] )
```

5.37.2.13 ldl_numeric()

```
int ldl_numeric (
            int n,
            int Ap[ ],
            int Ai[ ],
            abip_float Ax[ ],
            int Lp[ ],
            int Parent[ ],
            int Lnz[ ],
            int Li[ ],
            abip_float Lx[ ],
            abip_float D[ ],
            abip_float Y[ ],
            int Pattern[ ],
            int Flag[ ],
            int P[ ],
            int Pinv[ ] )
```

5.37.2.14 ldl_perm()

```
void ldl_perm (
            int n,
            abip_float X[ ],
            abip_float B[ ],
            int P[ ] )
```

5.37.2.15 ldl_permt()

```
void ldl_permt (
            int n,
            abip_float X[ ],
            abip_float B[ ],
            int P[ ] )
```

5.37.2.16 ldl_symbolic()

```
void ldl_symbolic (
            int n,
            int Ap[ ],
            int Ai[ ],
            int Lp[ ],
            int Parent[ ],
            int Lnz[ ],
            int Flag[ ],
            int P[ ],
            int Pinv[ ] )
```

5.37.2.17 ldl_valid_matrix()

```
int ldl_valid_matrix (
        int n,
        int Ap[ ],
        int Ai[ ] )
```

5.37.2.18 ldl_valid_perm()

```
int ldl_valid_perm (
        int n,
        int P[ ],
        int Flag[ ] )
```

5.38 ldl.h

Go to the documentation of this file.

```
/* ========================================================================== */
/* === ldl.h:  include file for the LDL package ============================= */
/* ========================================================================== */

/* Copyright (c) Timothy A Davis, http://www.suitesparse.com.
 * All Rights Reserved.  See LDL/Doc/License.txt for the License.
 */

#include "SuiteSparse_config.h"

#ifdef DLONG
#define LDL_int SuiteSparse_long
#define LDL_ID SuiteSparse_long_id

#define LDL_symbolic ldl_l_symbolic
#define LDL_numeric ldl_l_numeric
#define LDL_lsolve ldl_l_lsolve
#define LDL_dsolve ldl_l_dsolve
#define LDL_ltsolve ldl_l_ltsolve
#define LDL_perm ldl_l_perm
#define LDL_permt ldl_l_permt
#define LDL_valid_perm ldl_l_valid_perm
#define LDL_valid_matrix ldl_l_valid_matrix

#else
#define LDL_int int
#define LDL_ID "%d"

#define LDL_symbolic ldl_symbolic
#define LDL_numeric ldl_numeric
#define LDL_lsolve ldl_lsolve
#define LDL_dsolve ldl_dsolve
#define LDL_ltsolve ldl_ltsolve
#define LDL_perm ldl_perm
#define LDL_permt ldl_permt
#define LDL_valid_perm ldl_valid_perm
#define LDL_valid_matrix ldl_valid_matrix

#endif

/* ========================================================================== */
/* === int version ========================================================== */
/* ========================================================================== */

void ldl_symbolic (int n, int Ap [ ], int Ai [ ], int Lp [ ],
    int Parent [ ], int Lnz [ ], int Flag [ ], int P [ ], int Pinv [ ]) ;

int ldl_numeric (int n, int Ap [ ], int Ai [ ], abip_float Ax [ ],
    int Lp [ ], int Parent [ ], int Lnz [ ], int Li [ ], abip_float Lx [ ],
    abip_float D [ ], abip_float Y [ ], int Pattern [ ], int Flag [ ],
    int P [ ], int Pinv [ ]) ;

```

```
void ldl_lsolve (int n, abip_float X [ ], int Lp [ ], int Li [ ],
    abip_float Lx [ ]) ;

void ldl_dsolve (int n, abip_float X [ ], abip_float D [ ]) ;

void ldl_ltsolve (int n, abip_float X [ ], int Lp [ ], int Li [ ],
    abip_float Lx [ ]) ;

void ldl_perm  (int n, abip_float X [ ], abip_float B [ ], int P [ ]) ;
void ldl_permt (int n, abip_float X [ ], abip_float B [ ], int P [ ]) ;

int ldl_valid_perm (int n, int P [ ], int Flag [ ]) ;
int ldl_valid_matrix ( int n, int Ap [ ], int Ai [ ]) ;

/* ========================================================================== */
/* === long version ========================================================= */
/* ========================================================================== */

void ldl_l_symbolic (SuiteSparse_long n, SuiteSparse_long Ap [ ],
    SuiteSparse_long Ai [ ], SuiteSparse_long Lp [ ],
    SuiteSparse_long Parent [ ], SuiteSparse_long Lnz [ ],
    SuiteSparse_long Flag [ ], SuiteSparse_long P [ ],
    SuiteSparse_long Pinv [ ]) ;

SuiteSparse_long ldl_l_numeric (SuiteSparse_long n, SuiteSparse_long Ap [ ],
    SuiteSparse_long Ai [ ], abip_float Ax [ ], SuiteSparse_long Lp [ ],
    SuiteSparse_long Parent [ ], SuiteSparse_long Lnz [ ],
    SuiteSparse_long Li [ ], abip_float Lx [ ], abip_float D [ ], abip_float Y [ ],
    SuiteSparse_long Pattern [ ], SuiteSparse_long Flag [ ],
    SuiteSparse_long P [ ], SuiteSparse_long Pinv [ ]) ;

void ldl_l_lsolve (SuiteSparse_long n, abip_float X [ ], SuiteSparse_long Lp [ ],
    SuiteSparse_long Li [ ], abip_float Lx [ ]) ;

void ldl_l_dsolve (SuiteSparse_long n, abip_float X [ ], abip_float D [ ]) ;

void ldl_l_ltsolve (SuiteSparse_long n, abip_float X [ ], SuiteSparse_long Lp [ ],
    SuiteSparse_long Li [ ], abip_float Lx [ ]) ;

void ldl_l_perm  (SuiteSparse_long n, abip_float X [ ], abip_float B [ ],
    SuiteSparse_long P [ ]) ;
void ldl_l_permt (SuiteSparse_long n, abip_float X [ ], abip_float B [ ],
    SuiteSparse_long P [ ]) ;

SuiteSparse_long ldl_l_valid_perm (SuiteSparse_long n, SuiteSparse_long P [ ],
    SuiteSparse_long Flag [ ]) ;
SuiteSparse_long ldl_l_valid_matrix ( SuiteSparse_long n,
    SuiteSparse_long Ap [ ], SuiteSparse_long Ai [ ]) ;

/* ========================================================================== */
/* === LDL version ========================================================== */
/* ========================================================================== */

#define LDL_DATE "May 4, 2016"
#define LDL_VERSION_CODE(main,sub) ((main) * 1000 + (sub))
#define LDL_MAIN_VERSION 2
#define LDL_SUB_VERSION 2
#define LDL_SUBSUB_VERSION 6
#define LDL_VERSION LDL_VERSION_CODE(LDL_MAIN_VERSION,LDL_SUB_VERSION)

```

5.39 external/README.md File Reference

5.40 external/SuiteSparse_config.c File Reference

```
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include "SuiteSparse_config.h"
```

Functions

- void SuiteSparse_start (void)

- void SuiteSparse_finish (void)
- void * SuiteSparse_malloc (size_t nitems, size_t size_of_item)
- void * SuiteSparse_calloc (size_t nitems, size_t size_of_item)
- void * SuiteSparse_realloc (size_t nitems_new, size_t nitems_old, size_t size_of_item, void *p, int *ok)
- void * SuiteSparse_free (void *p)
- void SuiteSparse_tic (abip_float tic[2])
- abip_float SuiteSparse_toc (abip_float tic[2])
- abip_float SuiteSparse_time (void)
- int SuiteSparse_version (int version[3])
- abip_float SuiteSparse_hypot (abip_float x, abip_float y)
- int SuiteSparse_divcomplex (abip_float ar, abip_float ai, abip_float br, abip_float bi, abip_float *cr, abip_float *ci)

Variables

- struct SuiteSparse_config_struct SuiteSparse_config

5.40.1 Function Documentation

5.40.1.1 SuiteSparse_calloc()

```
void * SuiteSparse_calloc (
            size_t nitems,
            size_t size_of_item )
```

Definition at line 211 of file SuiteSparse_config.c.

5.40.1.2 SuiteSparse_divcomplex()

```
int SuiteSparse_divcomplex (
            abip_float ar,
            abip_float ai,
            abip_float br,
            abip_float bi,
            abip_float * cr,
            abip_float * ci )
```

Definition at line 516 of file SuiteSparse_config.c.

5.40.1.3 SuiteSparse_finish()

```
void SuiteSparse_finish (
            void )
```

Definition at line 173 of file SuiteSparse_config.c.

5.40.1.4 SuiteSparse_free()

```
void * SuiteSparse_free (
            void * p )
```

Definition at line 311 of file SuiteSparse_config.c.

5.40.1.5 SuiteSparse_hypot()

```
abip_float SuiteSparse_hypot (
            abip_float x,
            abip_float y )
```

Definition at line 464 of file SuiteSparse_config.c.

5.40.1.6 SuiteSparse_malloc()

```
void * SuiteSparse_malloc (
            size_t nitems,
            size_t size_of_item )
```

Definition at line 182 of file SuiteSparse_config.c.

5.40.1.7 SuiteSparse_realloc()

```
void * SuiteSparse_realloc (
            size_t nitems_new,
            size_t nitems_old,
            size_t size_of_item,
            void * p,
            int * ok )
```

Definition at line 247 of file SuiteSparse_config.c.

5.40.1.8 SuiteSparse_start()

```
void SuiteSparse_start (
            void )
```

Definition at line 109 of file SuiteSparse_config.c.

5.40.1.9 SuiteSparse_tic()

```
void SuiteSparse_tic (
            abip_float tic[2] )
```

Definition at line 373 of file SuiteSparse_config.c.

5.40.1.10 SuiteSparse_time()

```
abip_float SuiteSparse_time (
            void )
```

Definition at line 414 of file SuiteSparse_config.c.

5.40.1.11 SuiteSparse_toc()

```
abip_float SuiteSparse_toc (
            abip_float tic[2] )
```

Definition at line 397 of file SuiteSparse_config.c.

5.40.1.12 SuiteSparse_version()

```
int SuiteSparse_version (
            int version[3] )
```

Definition at line 429 of file SuiteSparse_config.c.

5.40.2 Variable Documentation

5.40.2.1 SuiteSparse_config

```
struct SuiteSparse_config_struct SuiteSparse_config
```

Definition at line 53 of file SuiteSparse_config.c.

5.41 SuiteSparse_config.c

Go to the documentation of this file.

```
/* ========================================================================== */
/* === SuiteSparse_config =================================================== */
/* ========================================================================== */

/* SuiteSparse configuration : memory manager and printf functions. */

/* Copyright (c) 2013, Timothy A. Davis.  No licensing restrictions
 * apply to this file or to the SuiteSparse_config directory.
 * Author: Timothy A. Davis.
 */

#include <math.h>
#include <stdlib.h>

#ifndef NPRINT
#include <stdio.h>
#endif

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#include "matrix.h"
#endif

#ifdef ABIP_NULL
#undef ABIP_NULL
#define ABIP_NULL ((void *) 0)
#endif

#include "SuiteSparse_config.h"

/* -------------------------------------------------------------------------- */
/* SuiteSparse_config : a global extern struct */
/* -------------------------------------------------------------------------- */

/* The SuiteSparse_config struct is available to all SuiteSparse functions and
    to all applications that use those functions.  It must be modified with
    care, particularly in a multithreaded context.  Normally, the application
    will initialize this object once, via SuiteSparse_start, possibily followed
    by application-specific modifications if the applications wants to use
    alternative memory manager functions.

    The user can redefine these global pointers at run-time to change the
    memory manager and printf function used by SuiteSparse.

    If -DNMALLOC is defined at compile-time, then no memory-manager is
    specified.  You must define them at run-time, after calling
    SuiteSparse_start.

    If -DPRINT is defined a compile time, then printf is disabled, and
    SuiteSparse will not use printf.
 */

struct SuiteSparse_config_struct SuiteSparse_config =
{

            /* memory management functions */
            #ifndef NMALLOC

            #ifdef MATLAB_MEX_FILE
                        /* MATLAB mexFunction: */
                        mxMalloc, mxCalloc, mxRealloc, mxFree,
            #else
                        /* standard ANSI C: */
                        malloc, calloc, realloc, free,
            #endif

            #else
                        /* no memory manager defined; you must define one at run-time: */
                        ABIP_NULL, ABIP_NULL, ABIP_NULL, ABIP_NULL,
            #endif

            /* printf function */
            #ifndef NPRINT

            #ifdef MATLAB_MEX_FILE
                        /* MATLAB mexFunction: */
                        mexPrintf,
            #else
                        /* standard ANSI C: */
                        printf,
            #endif

```

```
            #else
                    /* printf is disabled */
                    ABIP_NULL,
            #endif

            SuiteSparse_hypot,
            SuiteSparse_divcomplex

} ;

/* -------------------------------------------------------------------------- */
/* SuiteSparse_start */
/* -------------------------------------------------------------------------- */

/* All applications that use SuiteSparse should call SuiteSparse_start prior
   to using any SuiteSparse function.  Only a single thread should call this
   function, in a multithreaded application.  Currently, this function is
   optional, since all this function currently does is to set the four memory
   function pointers to ABIP_NULL (which tells SuiteSparse to use the default
   functions).  In a multi- threaded application, only a single thread should
   call this function.

   Future releases of SuiteSparse might enforce a requirement that
   SuiteSparse_start be called prior to calling any SuiteSparse function.
 */

void SuiteSparse_start ( void )
{

            /* memory management functions */
            #ifndef NMALLOC

            #ifdef MATLAB_MEX_FILE
                    /* MATLAB mexFunction: */
                    SuiteSparse_config.malloc_func  = mxMalloc ;
                    SuiteSparse_config.calloc_func  = mxCalloc ;
                    SuiteSparse_config.realloc_func = mxRealloc ;
                    SuiteSparse_config.free_func    = mxFree ;
            #else
                    /* standard ANSI C: */
                    SuiteSparse_config.malloc_func  = malloc ;
                    SuiteSparse_config.calloc_func  = calloc ;
                    SuiteSparse_config.realloc_func = realloc ;
                    SuiteSparse_config.free_func    = free ;
            #endif

            #else
                    /* no memory manager defined; you must define one after calling
 SuiteSparse_start */
                    SuiteSparse_config.malloc_func  = ABIP_NULL ;
                    SuiteSparse_config.calloc_func  = ABIP_NULL ;
                    SuiteSparse_config.realloc_func = ABIP_NULL ;
                    SuiteSparse_config.free_func = ABIP_NULL ;
            #endif

            /* printf function */
            #ifndef NPRINT

            #ifdef MATLAB_MEX_FILE
                    /* MATLAB mexFunction: */
                    SuiteSparse_config.printf_func = mexPrintf ;
            #else
                    /* standard ANSI C: */
                    SuiteSparse_config.printf_func = printf ;
            #endif

            #else
                    /* printf is disabled */
                    SuiteSparse_config.printf_func = ABIP_NULL ;

            #endif

            /* math functions */
            SuiteSparse_config.hypot_func = SuiteSparse_hypot ;
            SuiteSparse_config.divcomplex_func = SuiteSparse_divcomplex ;
}

/* -------------------------------------------------------------------------- */
/* SuiteSparse_finish */
/* -------------------------------------------------------------------------- */

/* This currently does nothing, but in the future, applications should call
   SuiteSparse_start before calling any SuiteSparse function, and then
   SuiteSparse_finish after calling the last SuiteSparse function, just before
   exiting.  In a multithreaded application, only a single thread should call
   this function.

```

```
   Future releases of SuiteSparse might use this function for any
   SuiteSparse-wide cleanup operations or finalization of statistics.
 */

void SuiteSparse_finish ( void )
{
    /* do nothing */ ;
}

/* -------------------------------------------------------------------------- */
/* SuiteSparse_malloc: malloc wrapper */
/* -------------------------------------------------------------------------- */

void *SuiteSparse_malloc      /* pointer to allocated block of memory */
(
            size_t nitems,             /* number of items to malloc */
            size_t size_of_item     /* sizeof each item */
)
{
            void *p ;
            size_t size ;
            if (nitems < 1) nitems = 1 ;
            if (size_of_item < 1) size_of_item = 1 ;
            size = nitems * size_of_item  ;

            if (size != ((abip_float) nitems) * size_of_item)
            {
                        /* size_t overflow */
                        p = ABIP_NULL ;
            }
            else
            {
                        p = (void *) (SuiteSparse_config.malloc_func) (size) ;
            }
            return (p) ;
}


/* -------------------------------------------------------------------------- */
/* SuiteSparse_calloc: calloc wrapper */
/* -------------------------------------------------------------------------- */

void *SuiteSparse_calloc    /* pointer to allocated block of memory */
(
            size_t nitems,          /* number of items to calloc */
            size_t size_of_item     /* sizeof each item */
)
{
            void *p ;
            size_t size ;
            if (nitems < 1) nitems = 1 ;
            if (size_of_item < 1) size_of_item = 1 ;
            size = nitems * size_of_item  ;

            if (size != ((abip_float) nitems) * size_of_item)
            {
                        /* size_t overflow */
                        p = ABIP_NULL ;
            }
            else
            {
                        p = (void *) (SuiteSparse_config.calloc_func) (nitems, size_of_item) ;
            }
            return (p) ;
}

/* -------------------------------------------------------------------------- */
/* SuiteSparse_realloc: realloc wrapper */
/* -------------------------------------------------------------------------- */

/* If p is non-NULL on input, it points to a previously allocated object of
   size nitems_old * size_of_item.  The object is reallocated to be of size
   nitems_new * size_of_item.  If p is NULL on input, then a new object of that
   size is allocated.  On success, a pointer to the new object is returned,
   and ok is returned as 1.  If the allocation fails, ok is set to 0 and a
   pointer to the old (unmodified) object is returned.
 */

void *SuiteSparse_realloc   /* pointer to reallocated block of memory, or to original block if the
 realloc failed. */
(
            size_t nitems_new,      /* new number of items in the object */
            size_t nitems_old,      /* old number of items in the object */
            size_t size_of_item,    /* sizeof each item */
            void *p,                /* old object to reallocate */
            int *ok                 /* 1 if successful, 0 otherwise */
)
```

```
{
        size_t size ;
        if (nitems_old < 1) nitems_old = 1 ;
        if (nitems_new < 1) nitems_new = 1 ;
        if (size_of_item < 1) size_of_item = 1 ;
        size = nitems_new * size_of_item  ;

        if (size != ((abip_float) nitems_new) * size_of_item)
        {
                /* size_t overflow */
                (*ok) = 0 ;
        }
        else if (p == ABIP_NULL)
        {
                /* a fresh object is being allocated */
                p = SuiteSparse_malloc (nitems_new, size_of_item) ;
                (*ok) = (p != ABIP_NULL) ;
        }
        else if (nitems_old == nitems_new)
        {
                /* the object does not change; do nothing */
                (*ok) = 1 ;
        }
        else
        {
                /* change the size of the object from nitems_old to nitems_new */
                void *pnew ;
                pnew = (void *) (SuiteSparse_config.realloc_func) (p, size) ;

                if (pnew == ABIP_NULL)
                {
                        if (nitems_new < nitems_old)
                        {
                                /* the attempt to reduce the size of the block failed,
   but the old block is unchanged.  So pretend to succeed. */
                                (*ok) = 1 ;
                        }
                        else
                        {
                                /* out of memory */
                                (*ok) = 0 ;
                        }
                }
                else
                {
                        /* success */
                        p = pnew ;
                        (*ok) = 1 ;
                }
        }
        return (p) ;
}

/* ------------------------------------------------------------------------- */
/* SuiteSparse_free: free wrapper */
/* ------------------------------------------------------------------------- */

void *SuiteSparse_free      /* always returns ABIP_NULL */
(
        void *p                 /* block to free */
)
{
        if (p)
        {
                (SuiteSparse_config.free_func) (p) ;
        }
        return (ABIP_NULL) ;
}


/* ------------------------------------------------------------------------- */
/* SuiteSparse_tic: return current wall clock time */
/* ------------------------------------------------------------------------- */

/* Returns the number of seconds (tic [0]) and nanoseconds (tic [1]) since some
 * unspecified but fixed time in the past.  If no timer is installed, zero is
 * returned.  A scalar abip_float precision value for 'tic' could be used, but this
 * might cause loss of precision because clock_getttime returns the time from
 * some distant time in the past.  Thus, an array of size 2 is used.
 *
 * The timer is enabled by default.  To disable the timer, compile with
 * -DNTIMER.  If enabled on a POSIX C 1993 system, the timer requires linking
 * with the -lrt library.
 *
 * example:
 *
 *      abip_float tic [2], r, s, t ;
```

```
 *      SuiteSparse_tic (tic) ;     // start the timer
 *      // do some work A
 *      t = SuiteSparse_toc (tic) ; // t is time for work A, in seconds
 *      // do some work B
 *      s = SuiteSparse_toc (tic) ; // s is time for work A and B, in seconds
 *      SuiteSparse_tic (tic) ;     // restart the timer
 *      // do some work C
 *      r = SuiteSparse_toc (tic) ; // s is time for work C, in seconds
 *
 * A abip_float array of size 2 is used so that this routine can be more easily
 * ported to non-POSIX systems.  The caller does not rely on the POSIX
 * <time.h> include file.
 */

#ifdef SUITESPARSE_TIMER_ENABLED

#include <time.h>

void SuiteSparse_tic
(
            abip_float tic [2]      /* output, contents undefined on input */
)
{
            /* POSIX C 1993 timer, requires -librt */
            struct timespec t ;
            clock_gettime (CLOCK_MONOTONIC, &t) ;
            tic [0] = (abip_float) (t.tv_sec) ;
            tic [1] = (abip_float) (t.tv_nsec) ;
}

#else

void SuiteSparse_tic
(
            abip_float tic [2]      /* output, contents undefined on input */
)
{
            /* no timer installed */
            tic [0] = 0 ;
            tic [1] = 0 ;
}

#endif


/* -------------------------------------------------------------------------- */
/* SuiteSparse_toc: return time since last tic */
/* -------------------------------------------------------------------------- */

/* Assuming SuiteSparse_tic is accurate to the nanosecond, this function is
 * accurate down to the nanosecond for 2^53 nanoseconds since the last call to
 * SuiteSparse_tic, which is sufficient for SuiteSparse (about 104 days).  If
 * additional accuracy is required, the caller can use two calls to
 * SuiteSparse_tic and do the calculations differently.
 */

abip_float SuiteSparse_toc  /* returns time in seconds since last tic */
(
            abip_float tic [2]  /* input, not modified from last call to SuiteSparse_tic */
)
{
            abip_float toc [2] ;
            SuiteSparse_tic (toc) ;
            return ((toc [0] - tic [0]) + 1e-9 * (toc [1] - tic [1])) ;
}


/* -------------------------------------------------------------------------- */
/* SuiteSparse_time: return current wallclock time in seconds */
/* -------------------------------------------------------------------------- */

/* This function might not be accurate down to the nanosecond. */

abip_float SuiteSparse_time  /* returns current wall clock time in seconds */
(
            void
)
{
            abip_float toc [2] ;
            SuiteSparse_tic (toc) ;
            return (toc [0] + 1e-9 * toc [1]) ;
}


/* -------------------------------------------------------------------------- */
/* SuiteSparse_version: return the current version of SuiteSparse */
/* -------------------------------------------------------------------------- */
```

```
int SuiteSparse_version
(
            int version [3]
)
{
            if (version != ABIP_NULL)
            {
                        version [0] = SUITESPARSE_MAIN_VERSION ;
                        version [1] = SUITESPARSE_SUB_VERSION ;
                        version [2] = SUITESPARSE_SUBSUB_VERSION ;
            }
            return (SUITESPARSE_VERSION) ;
}

/* -------------------------------------------------------------------------- */
/* SuiteSparse_hypot */
/* -------------------------------------------------------------------------- */

/* There is an equivalent routine called hypot in <math.h>, which conforms
 * to ANSI C99.  However, SuiteSparse does not assume that ANSI C99 is
 * available.  You can use the ANSI C99 hypot routine with:
 *
 *      #include <math.h>
 *i     SuiteSparse_config.hypot_func = hypot ;
 *
 * Default value of the SuiteSparse_config.hypot_func pointer is
 * SuiteSparse_hypot, defined below.
 *
 * s = hypot (x,y) computes s = sqrt (x*x + y*y) but does so more accurately.
 * The NaN cases for the abip_float relops x >= y and x+y == x are safely ignored.
 *
 * Source: Algorithm 312, "Absolute value and square root of a complex number,"
 * P. Friedland, Comm. ACM, vol 10, no 10, October 1967, page 665.
 */

abip_float SuiteSparse_hypot (abip_float x, abip_float y)
{
            abip_float s;
            abip_float r ;
            x = fabs (x) ;
            y = fabs (y) ;

            if (x >= y)
            {
                        if (x + y == x)
                        {
                                    s = x ;
                        }
                        else
                        {
                                    r = y / x ;
                                    s = x * sqrt (1.0 + r*r) ;
                        }
            }
            else
            {
                        if (y + x == y)
                        {
                                    s = y ;
                        }
                        else
                        {
                                    r = x / y ;
                                    s = y * sqrt (1.0 + r*r) ;
                        }
            }
            return (s) ;
}

/* -------------------------------------------------------------------------- */
/* SuiteSparse_divcomplex */
/* -------------------------------------------------------------------------- */

/* c = a/b where c, a, and b are complex.  The real and imaginary parts are
 * passed as separate arguments to this routine.  The NaN case is ignored
 * for the abip_float relop br >= bi.  Returns 1 if the denominator is zero,
 * 0 otherwise.
 *
 * This uses ACM Algo 116, by R. L. Smith, 1962, which tries to avoid
 * underflow and overflow.
 *
 * c can be the same variable as a or b.
 *
 * Default value of the SuiteSparse_config.divcomplex_func pointer is
 * SuiteSparse_divcomplex.
 */
```

```c

int SuiteSparse_divcomplex
(
            abip_float ar,
            abip_float ai,       /* real and imaginary parts of a */
            abip_float br,
            abip_float bi,       /* real and imaginary parts of b */
            abip_float *cr,
            abip_float *ci      /* real and imaginary parts of c */
)
{
            abip_float tr;
            abip_float ti;
            abip_float r;
            abip_float den;

            if (fabs (br) >= fabs (bi))
            {
                        r = bi / br ;
                        den = br + r * bi ;
                        tr = (ar + ai * r) / den ;
                        ti = (ai - ar * r) / den ;
            }
            else
            {
                        r = br / bi ;
                        den = r * br + bi ;
                        tr = (ar * r + ai) / den ;
                        ti = (ai * r - ar) / den ;
            }

            *cr = tr ;
            *ci = ti ;
            return (den == 0.) ;
}
```

5.42 external/SuiteSparse_config.h File Reference

```c
#include "glbopts.h"
#include "ctrlc.h"
#include <limits.h>
#include <stdlib.h>
```

Data Structures

- struct SuiteSparse_config_struct

Macros

- #define SuiteSparse_long long
- #define SuiteSparse_long_max LONG_MAX
- #define SuiteSparse_long_idd "ld"
- #define SuiteSparse_long_id "%" SuiteSparse_long_idd
- #define SUITESPARSE_PRINTF(params) {if (SuiteSparse_config.printf_func != ABIP_NULL){(void) (Suite↩ Sparse_config.printf_func) params;}}
- #define SUITESPARSE_HAS_VERSION_FUNCTION
- #define SUITESPARSE_DATE "Dec 28, 2017"
- #define SUITESPARSE_VER_CODE(main, sub) ((main) ∗ 1000 + (sub))
- #define SUITESPARSE_MAIN_VERSION 5
- #define SUITESPARSE_SUB_VERSION 1
- #define SUITESPARSE_SUBSUB_VERSION 2
- #define SUITESPARSE_VERSION SUITESPARSE_VER_CODE(SUITESPARSE_MAIN_VERSION,SUITESPARSE_SUB_VE

Functions

- void SuiteSparse_start (void)
- void SuiteSparse_finish (void)
- void * SuiteSparse_malloc (size_t nitems, size_t size_of_item)
- void * SuiteSparse_calloc (size_t nitems, size_t size_of_item)
- void * SuiteSparse_realloc (size_t nitems_new, size_t nitems_old, size_t size_of_item, void *p, int *ok)
- void * SuiteSparse_free (void *p)
- void SuiteSparse_tic (abip_float tic[2])
- abip_float SuiteSparse_toc (abip_float tic[2])
- abip_float SuiteSparse_time (void)
- abip_float SuiteSparse_hypot (abip_float x, abip_float y)
- int SuiteSparse_divcomplex (abip_float ar, abip_float ai, abip_float br, abip_float bi, abip_float *cr, abip_float *ci)
- int SuiteSparse_version (int version[3])

Variables

- struct SuiteSparse_config_struct SuiteSparse_config

5.42.1 Macro Definition Documentation

5.42.1.1 SUITESPARSE_DATE

```
#define SUITESPARSE_DATE "Dec 28, 2017"
```

Definition at line 237 of file SuiteSparse_config.h.

5.42.1.2 SUITESPARSE_HAS_VERSION_FUNCTION

```
#define SUITESPARSE_HAS_VERSION_FUNCTION
```

Definition at line 235 of file SuiteSparse_config.h.

5.42.1.3 SuiteSparse_long

```
#define SuiteSparse_long long
```

Definition at line 64 of file SuiteSparse_config.h.

5.42.1.4 SuiteSparse_long_id

```
#define SuiteSparse_long_id "%" SuiteSparse_long_idd
```

Definition at line 69 of file SuiteSparse_config.h.

5.42.1.5 SuiteSparse_long_idd

```
#define SuiteSparse_long_idd "ld"
```

Definition at line 66 of file SuiteSparse_config.h.

5.42.1.6 SuiteSparse_long_max

```
#define SuiteSparse_long_max LONG_MAX
```

Definition at line 65 of file SuiteSparse_config.h.

5.42.1.7 SUITESPARSE_MAIN_VERSION

```
#define SUITESPARSE_MAIN_VERSION 5
```

Definition at line 239 of file SuiteSparse_config.h.

5.42.1.8 SUITESPARSE_PRINTF

```
#define SUITESPARSE_PRINTF(
             params ) {if (SuiteSparse_config.printf_func != ABIP_NULL){(void) (SuiteSparse_←
config.printf_func) params;}}
```

Definition at line 170 of file SuiteSparse_config.h.

5.42.1.9 SUITESPARSE_SUB_VERSION

```
#define SUITESPARSE_SUB_VERSION 1
```

Definition at line 240 of file SuiteSparse_config.h.

5.42.1.10 SUITESPARSE_SUBSUB_VERSION

```
#define SUITESPARSE_SUBSUB_VERSION 2
```

Definition at line 241 of file SuiteSparse_config.h.

5.42.1.11 SUITESPARSE_VER_CODE

```
#define SUITESPARSE_VER_CODE(
            main,
            sub ) ((main) * 1000 + (sub))
```

Definition at line 238 of file SuiteSparse_config.h.

5.42.1.12 SUITESPARSE_VERSION

```
#define SUITESPARSE_VERSION  SUITESPARSE_VER_CODE(SUITESPARSE_MAIN_VERSION,SUITESPARSE_SUB_VERSION)
```

Definition at line 242 of file SuiteSparse_config.h.

5.42.2 Function Documentation

5.42.2.1 SuiteSparse_calloc()

```
void * SuiteSparse_calloc (
            size_t nitems,
            size_t size_of_item )
```

Definition at line 211 of file SuiteSparse_config.c.

5.42.2.2 SuiteSparse_divcomplex()

```
int SuiteSparse_divcomplex (
            abip_float ar,
            abip_float ai,
            abip_float br,
            abip_float bi,
            abip_float * cr,
            abip_float * ci )
```

Definition at line 516 of file SuiteSparse_config.c.

5.42.2.3 SuiteSparse_finish()

```
void SuiteSparse_finish (
            void )
```

Definition at line 173 of file SuiteSparse_config.c.

5.42.2.4 SuiteSparse_free()

```
void * SuiteSparse_free (
            void * p )
```

Definition at line 311 of file SuiteSparse_config.c.

5.42.2.5 SuiteSparse_hypot()

```
abip_float SuiteSparse_hypot (
            abip_float x,
            abip_float y )
```

Definition at line 464 of file SuiteSparse_config.c.

5.42.2.6 SuiteSparse_malloc()

```
void * SuiteSparse_malloc (
            size_t nitems,
            size_t size_of_item )
```

Definition at line 182 of file SuiteSparse_config.c.

5.42.2.7 SuiteSparse_realloc()

```
void * SuiteSparse_realloc (
            size_t nitems_new,
            size_t nitems_old,
            size_t size_of_item,
            void * p,
            int * ok )
```

Definition at line 247 of file SuiteSparse_config.c.

5.42.2.8 SuiteSparse_start()

```
void SuiteSparse_start (
            void )
```

Definition at line 109 of file SuiteSparse_config.c.

5.42.2.9 SuiteSparse_tic()

```
void SuiteSparse_tic (
            abip_float tic[2] )
```

Definition at line 373 of file SuiteSparse_config.c.

5.42.2.10 SuiteSparse_time()

```
abip_float SuiteSparse_time (
            void )
```

Definition at line 414 of file SuiteSparse_config.c.

5.42.2.11 SuiteSparse_toc()

```
abip_float SuiteSparse_toc (
            abip_float tic[2] )
```

Definition at line 397 of file SuiteSparse_config.c.

5.42.2.12 SuiteSparse_version()

```
int SuiteSparse_version (
            int version[3] )
```

Definition at line 429 of file SuiteSparse_config.c.

5.42.3 Variable Documentation

5.42.3.1 SuiteSparse_config

```
struct SuiteSparse_config_struct SuiteSparse_config [extern]
```

Definition at line 53 of file SuiteSparse_config.c.

5.43 SuiteSparse_config.h

Go to the documentation of this file.

```
/* ========================================================================== */
/* === SuiteSparse_config =================================================== */
/* ========================================================================== */

/* Configuration file for SuiteSparse: a Suite of Sparse matrix packages
 * (AMD, COLAMD, CCOLAMD, CAMD, CHOLMOD, UMFPACK, CXSparse, and others).
 *
 * SuiteSparse_config.h provides the definition of the long integer.  On most
 * systems, a C program can be compiled in LP64 mode, in which long's and
 * pointers are both 64-bits, and int's are 32-bits.  Windows 64, however, uses
 * the LLP64 model, in which int's and long's are 32-bits, and long long's and
 * pointers are 64-bits.
 *
 * SuiteSparse packages that include long integer versions are
 * intended for the LP64 mode.  However, as a workaround for Windows 64
 * (and perhaps other systems), the long integer can be redefined.
 *
 * If _WIN64 is defined, then the __int64 type is used instead of long.
 *
 * The long integer can also be defined at compile time.  For example, this
 * could be added to SuiteSparse_config.mk:
 *
 * CFLAGS = -O -D'SuiteSparse_long=long long' \
 *  -D'SuiteSparse_long_max=9223372036854775801' -D'SuiteSparse_long_idd="lld"'
 *
 * This file defines SuiteSparse_long as either long (on all but _WIN64) or
 * __int64 on Windows 64.  The intent is that a SuiteSparse_long is always a
 * 64-bit integer in a 64-bit code.  ptrdiff_t might be a better choice than
 * long; it is always the same size as a pointer.
 *
 * This file also defines the SUITESPARSE_VERSION and related definitions.
 *
 * Copyright (c) 2012, Timothy A. Davis.  No licensing restrictions apply
 * to this file or to the SuiteSparse_config directory.
 * Author: Timothy A. Davis.
 */

#ifndef SUITESPARSE_CONFIG_H
#define SUITESPARSE_CONFIG_H

#ifdef __cplusplus
extern "C" {
#endif

#include "glbopts.h"
#include "ctrlc.h"
#include <limits.h>
#include <stdlib.h>

/* ========================================================================== */
/* === SuiteSparse_long ===================================================== */
/* ========================================================================== */

#ifndef SuiteSparse_long

#ifdef _WIN64

#define SuiteSparse_long __int64
#define SuiteSparse_long_max _I64_MAX
#define SuiteSparse_long_idd "I64d"

#else

#define SuiteSparse_long long
#define SuiteSparse_long_max LONG_MAX
#define SuiteSparse_long_idd "ld"

#endif
#define SuiteSparse_long_id "%" SuiteSparse_long_idd
```

```
#endif

/* ========================================================================== */
/* === SuiteSparse_config parameters and functions ========================== */
/* ========================================================================== */

/* SuiteSparse-wide parameters are placed in this struct.  It is meant to be
   an extern, globally-accessible struct.  It is not meant to be updated
   frequently by multiple threads.  Rather, if an application needs to modify
   SuiteSparse_config, it should do it once at the beginning of the application,
   before multiple threads are launched.

   The intent of these function pointers is that they not be used in your
   application directly, except to assign them to the desired user-provided
   functions.  Rather, you should use the
 */

struct SuiteSparse_config_struct
{
    void *(*malloc_func) (size_t) ;             /* pointer to malloc */
    void *(*calloc_func) (size_t, size_t) ;     /* pointer to calloc */
    void *(*realloc_func) (void *, size_t) ;    /* pointer to realloc */
    void (*free_func) (void *) ;                /* pointer to free */
    int (*printf_func) (const char *, ...) ;    /* pointer to printf */
    abip_float (*hypot_func) (abip_float, abip_float) ;     /* pointer to hypot */
    int (*divcomplex_func) (abip_float, abip_float, abip_float, abip_float, abip_float *, abip_float
 *);
} ;

extern struct SuiteSparse_config_struct SuiteSparse_config ;

void SuiteSparse_start ( void ) ;   /* called to start SuiteSparse */

void SuiteSparse_finish ( void ) ;  /* called to finish SuiteSparse */

void *SuiteSparse_malloc    /* pointer to allocated block of memory */
(
      size_t nitems,          /* number of items to malloc (>=1 is enforced) */
      size_t size_of_item     /* sizeof each item */
) ;

void *SuiteSparse_calloc    /* pointer to allocated block of memory */
(
      size_t nitems,          /* number of items to calloc (>=1 is enforced) */
      size_t size_of_item     /* sizeof each item */
) ;

void *SuiteSparse_realloc   /* pointer to reallocated block of memory, or
                               to original block if the realloc failed. */
(
      size_t nitems_new,      /* new number of items in the object */
      size_t nitems_old,      /* old number of items in the object */
      size_t size_of_item,    /* sizeof each item */
      void *p,                /* old object to reallocate */
      int *ok                 /* 1 if successful, 0 otherwise */
) ;

void *SuiteSparse_free      /* always returns NULL */
(
      void *p                 /* block to free */
) ;

void SuiteSparse_tic    /* start the timer */
(
      abip_float tic [2]      /* output, contents undefined on input */
) ;

abip_float SuiteSparse_toc  /* return time in seconds since last tic */
(
      abip_float tic [2]      /* input: from last call to SuiteSparse_tic */
) ;

abip_float SuiteSparse_time  /* returns current wall clock time in seconds */
(
    void
) ;

/* returns sqrt (x^2 + y^2), computed reliably */
abip_float SuiteSparse_hypot (abip_float x, abip_float y) ;

/* complex division of c = a/b */
int SuiteSparse_divcomplex
(
      abip_float ar,
      abip_float ai,     /* real and imaginary parts of a */
      abip_float br,
      abip_float bi,  /* real and imaginary parts of b */
```

```
      abip_float *cr,
      abip_float *ci  /* real and imaginary parts of c */
) ;

/* determine which timer to use, if any */
#ifndef NTIMER
#ifdef _POSIX_C_SOURCE
#if    _POSIX_C_SOURCE >= 199309L
#define SUITESPARSE_TIMER_ENABLED
#endif
#endif
#endif

/* SuiteSparse printf macro */
#define SUITESPARSE_PRINTF(params) {if (SuiteSparse_config.printf_func != ABIP_NULL){(void)
  (SuiteSparse_config.printf_func) params;}}

/* ========================================================================== */
/* === SuiteSparse version ================================================== */
/* ========================================================================== */

/* SuiteSparse is not a package itself, but a collection of packages, some of
 * which must be used together (UMFPACK requires AMD, CHOLMOD requires AMD,
 * COLAMD, CAMD, and CCOLAMD, etc).  A version number is provided here for the
 * collection itself.  The versions of packages within each version of
 * SuiteSparse are meant to work together.  Combining one package from one
 * version of SuiteSparse, with another package from another version of
 * SuiteSparse, may or may not work.
 *
 * SuiteSparse contains the following packages:
 *
 *  SuiteSparse_config version 5.1.2 (version always the same as SuiteSparse)
 *  GraphBLAS        version 1.1.2
 *  ssget            version 2.0.0
 *  AMD              version 2.4.6
 *  BTF              version 1.2.6
 *  CAMD             version 2.4.6
 *  CCOLAMD          version 2.9.6
 *  CHOLMOD          version 3.0.11
 *  COLAMD           version 2.9.6
 *  CSparse          version 3.2.0
 *  CXSparse         version 3.2.0
 *  GPUQREngine      version 1.0.5
 *  KLU              version 1.3.8
 *  LDL              version 2.2.6
 *  RBio             version 2.2.6
 *  SPQR             version 2.0.8
 *  SuiteSparse_GPURuntime  version 1.0.5
 *  UMFPACK          version 5.7.6
 *  MATLAB_Tools     various packages & M-files
 *  xerbla           version 1.0.3
 *
 * Other package dependencies:
 *  BLAS             required by CHOLMOD and UMFPACK
 *  LAPACK           required by CHOLMOD
 *  METIS 5.1.0      required by CHOLMOD (optional) and KLU (optional)
 *  CUBLAS, CUDART   NVIDIA libraries required by CHOLMOD and SPQR when
 *                   they are compiled with GPU acceleration.
 */

int SuiteSparse_version     /* returns SUITESPARSE_VERSION */
(
    /* output, not defined on input.  Not used if NULL.  Returns
       the three version codes in version [0..2]:
       version [0] is SUITESPARSE_MAIN_VERSION
       version [1] is SUITESPARSE_SUB_VERSION
       version [2] is SUITESPARSE_SUBSUB_VERSION
       */
    int version [3]
) ;

/* Versions prior to 4.2.0 do not have the above function.  The following
   code fragment will work with any version of SuiteSparse:

   #ifdef SUITESPARSE_HAS_VERSION_FUNCTION
   v = SuiteSparse_version (NULL) ;
   #else
   v = SUITESPARSE_VERSION ;
   #endif
*/
#define SUITESPARSE_HAS_VERSION_FUNCTION

#define SUITESPARSE_DATE "Dec 28, 2017"
#define SUITESPARSE_VER_CODE(main,sub) ((main) * 1000 + (sub))
#define SUITESPARSE_MAIN_VERSION 5
#define SUITESPARSE_SUB_VERSION 1
#define SUITESPARSE_SUBSUB_VERSION 2
```

```
#define SUITESPARSE_VERSION \
    SUITESPARSE_VER_CODE(SUITESPARSE_MAIN_VERSION,SUITESPARSE_SUB_VERSION)

#ifdef __cplusplus
}
#endif
#endif
```

5.44 include/abip.h File Reference

```
#include "glbopts.h"
#include <string.h>
```

Data Structures

- struct ABIP_PROBLEM_DATA
- struct ABIP_SETTINGS
- struct ABIP_SOL_VARS
- struct ABIP_INFO
- struct ABIP_SCALING
- struct ABIP_WORK
- struct ABIP_RESIDUALS

Typedefs

- typedef struct ABIP_A_DATA_MATRIX ABIPMatrix
- typedef struct ABIP_LIN_SYS_WORK ABIPLinSysWork
- typedef struct ABIP_PROBLEM_DATA ABIPData
- typedef struct ABIP_SETTINGS ABIPSettings
- typedef struct ABIP_SOL_VARS ABIPSolution
- typedef struct ABIP_INFO ABIPInfo
- typedef struct ABIP_SCALING ABIPScaling
- typedef struct ABIP_WORK ABIPWork
- typedef struct ABIP_ADAPTIVE_WORK ABIPAdaptWork
- typedef struct ABIP_RESIDUALS ABIPResiduals

Functions

- ABIPWork ∗ABIP() init (const ABIPData ∗d, ABIPInfo ∗info)
- abip_int ABIP() solve (ABIPWork ∗w, const ABIPData ∗d, ABIPSolution ∗sol, ABIPInfo ∗info)

  detailed update rule of ABIP
- void ABIP() finish (ABIPWork ∗w)
- abip_int ABIP() main (const ABIPData ∗d, ABIPSolution ∗sol, ABIPInfo ∗info)

  the main function
- const char ∗ABIP() version (void)

  return the abip version

5.44.1 Typedef Documentation

5.44.1.1 ABIPAdaptWork

```
typedef struct ABIP_ADAPTIVE_WORK ABIPAdaptWork
```

Definition at line 20 of file abip.h.

5.44.1.2 ABIPData

```
typedef struct ABIP_PROBLEM_DATA ABIPData
```

Definition at line 14 of file abip.h.

5.44.1.3 ABIPInfo

```
typedef struct ABIP_INFO ABIPInfo
```

Definition at line 17 of file abip.h.

5.44.1.4 ABIPLinSysWork

```
typedef struct ABIP_LIN_SYS_WORK ABIPLinSysWork
```

Definition at line 12 of file abip.h.

5.44.1.5 ABIPMatrix

```
typedef struct ABIP_A_DATA_MATRIX ABIPMatrix
```

Definition at line 11 of file abip.h.

5.44.1.6 ABIPResiduals

```
typedef struct ABIP_RESIDUALS ABIPResiduals
```

Definition at line 21 of file abip.h.

5.44.1.7 ABIPScaling

```
typedef struct ABIP_SCALING ABIPScaling
```

Definition at line 18 of file abip.h.

5.44.1.8 ABIPSettings

```
typedef struct ABIP_SETTINGS ABIPSettings
```

Definition at line 15 of file abip.h.

5.44.1.9 ABIPSolution

```
typedef struct ABIP_SOL_VARS ABIPSolution
```

Definition at line 16 of file abip.h.

5.44.1.10 ABIPWork

```
typedef struct ABIP_WORK ABIPWork
```

Definition at line 19 of file abip.h.

5.44.2 Function Documentation

5.44.2.1 finish()

```
void ABIP() finish (
             ABIPWork * w )
```

Definition at line 2301 of file abip.c.

5.44.2.2 init()

```
ABIPWork *ABIP() init (
             const ABIPData * d,
             ABIPInfo * info )
```

Definition at line 2341 of file abip.c.

5.44.2.3 main()

```
abip_int ABIP() main (
             const ABIPData * d,
             ABIPSolution * sol,
             ABIPInfo * info )
```

the main function

Definition at line 2393 of file abip.c.

5.44.2.4 solve()

```
abip_int ABIP() solve (
             ABIPWork * w,
             const ABIPData * d,
             ABIPSolution * sol,
             ABIPInfo * info )
```

detailed update rule of ABIP

Definition at line 2056 of file abip.c.

5.44.2.5 version()

```
const char *ABIP() version (
             void )
```

return the abip version

Definition at line 5 of file abip_version.c.

5.45 abip.h

Go to the documentation of this file.

```
#ifndef ABIP_H_GUARD
#define ABIP_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "glbopts.h"
#include <string.h>

typedef struct ABIP_A_DATA_MATRIX ABIPMatrix;
typedef struct ABIP_LIN_SYS_WORK ABIPLinSysWork;

typedef struct ABIP_PROBLEM_DATA ABIPData;
typedef struct ABIP_SETTINGS ABIPSettings;
typedef struct ABIP_SOL_VARS ABIPSolution;
typedef struct ABIP_INFO ABIPInfo;
typedef struct ABIP_SCALING ABIPScaling;
typedef struct ABIP_WORK ABIPWork;
typedef struct ABIP_ADAPTIVE_WORK ABIPAdaptWork;
typedef struct ABIP_RESIDUALS ABIPResiduals;

struct ABIP_PROBLEM_DATA
{
      abip_int m;
      abip_int n;
      ABIPMatrix *A;

      abip_float *b;
      abip_float *c;
      abip_float sp;

      ABIPSettings *stgs;
};

struct ABIP_SETTINGS
{
      abip_int normalize;
      abip_int pfeasopt;
      abip_float scale;
      abip_float rho_y;
      abip_float sparsity_ratio;

      abip_int max_ipm_iters;
      abip_int max_admm_iters;
      abip_float max_time;

      abip_float eps;
      abip_float alpha;
      abip_float cg_rate;  /* for indirect, tolerance goes down like (1/iter)^cg_rate: 2 */


      abip_int adaptive;
      abip_float eps_cor;
      abip_float eps_pen;

      abip_float dynamic_sigma; /* Dynamic strategy A to update the barrier parameter */

      abip_float dynamic_x; /* Dynamic strategy B to update the barrier parameter */
      abip_float dynamic_eta;

      abip_int   restart_fre; /* Frequency of performing restart*/
      abip_int   restart_thresh; /* Threshold of restart */

      abip_int verbose;    /* boolean, write out progress: 1 */
      abip_int warm_start; /* boolean, warm start (put initial guess in ABIPSolution struct): 0 */


      abip_int adaptive_lookback;


      abip_int origin_rescale;            /* boolean, use the origin_rescale or not */
      abip_int pc_ruiz_rescale;           /* boolean, use the pc_rescale or not */
      abip_int qp_rescale;                /* boolean, use the qp rescale or not*/
      abip_int ruiz_iter;                 /* int, number of ruiz rescaling */
      abip_int hybrid_mu;                 /* boolean, use the hybrid_mu strategy or not */
      abip_float hybrid_thresh;             /* float, control when to use the LOQO mu strategy, only used when hybrid_mu = 1 */
      abip_float dynamic_sigma_second;    /* float, control the dynamic sigma when the mu strategy becomes LOQO */
      abip_int half_update;              /* boolean, use the half update technique or not*/
      abip_int avg_criterion;           /*  boolean, use the avg criterion technique or not*/
```

```
};

struct ABIP_SOL_VARS
{
      abip_float *x;
      abip_float *y;
      abip_float *s;
};

struct ABIP_INFO
{
      char status[32];
      abip_int status_val;
      abip_int ipm_iter;
      abip_int admm_iter;

      abip_float pobj;
      abip_float dobj;
      abip_float res_pri;
      abip_float res_dual;
      abip_float rel_gap;
      abip_float res_infeas;
      abip_float res_unbdd;

      abip_float setup_time;
      abip_float solve_time;
};

struct ABIP_SCALING
{
      abip_float *D;
      abip_float *E;

      abip_float mean_norm_row_A;
      abip_float mean_norm_col_A;
};

/* main functions: ABIP(init): allocates memory of matrix, e.g., [I A; A^T -I].
                                ABIP(solve): can be called many times with different b,c data per init call.
                                ABIP(finish): cleans up the memory (one per init call) */
ABIPWork *ABIP(init)(const ABIPData *d, ABIPInfo *info);
abip_int ABIP(solve)(ABIPWork *w, const ABIPData *d, ABIPSolution *sol, ABIPInfo *info);
void ABIP(finish)(ABIPWork *w);

abip_int ABIP(main)(const ABIPData *d, ABIPSolution *sol, ABIPInfo *info);
const char *ABIP(version)(void);

struct ABIP_WORK
{
      abip_float sigma;
      abip_float gamma;
      abip_int final_check;
      abip_int double_check;

      abip_float mu;
      abip_float beta;

      abip_float *u;
      abip_float *v;
      abip_float *u_t;
      abip_float *u_prev;
      abip_float *v_prev;

      abip_float* u_avg;
      abip_float* v_avg;

      abip_float* u_avgcon;
      abip_float* v_avgcon;

      abip_float* u_sumcon;
      abip_float* v_sumcon;


      abip_int    fre_old;

      abip_float *h;
      abip_float *g;
      abip_float *pr;
      abip_float *dr;

      abip_float g_th;
      abip_float sc_b;
      abip_float sc_c;
      abip_float nm_b;
      abip_float nm_c;

```

```
    abip_float *b;
    abip_float *c;
    abip_int m;
    abip_int n;
    ABIPMatrix *A;
    abip_float sp;

    ABIPLinSysWork *p;
    ABIPAdaptWork *adapt;
    ABIPSettings *stgs;
    ABIPScaling *scal;
};

struct ABIP_RESIDUALS
{
    abip_int last_ipm_iter;
    abip_int last_admm_iter;
    abip_float last_mu;

    abip_float res_pri;
    abip_float res_dual;
    abip_float rel_gap;
    abip_float res_infeas;
    abip_float res_unbdd;

    abip_float ct_x_by_tau;
    abip_float bt_y_by_tau;

    abip_float tau;
    abip_float kap;
};

#ifdef __cplusplus
}
#endif
#endif
```

5.46 include/abip_blas.h File Reference

5.47 abip_blas.h

Go to the documentation of this file.

```
#ifndef ABIP_BLAS_H_GUARD
#define ABIP_BLAS_H_GUARD

#ifdef USE_LAPACK

#ifdef __cplusplus
extern "C" {
#endif

#ifndef BLASSUFFIX
#define BLASSUFFIX _
#endif

#if defined(NOBLASSUFFIX) && NOBLASSUFFIX > 0
#ifndef SFLOAT
#define BLAS(x) d##x
#else
#define BLAS(x) s##x
#endif
#else
#define stitch_(pre, x, post) pre##x##post
#define stitch__(pre, x, post) stitch_(pre, x, post)
#ifndef SFLOAT
#define BLAS(x) stitch__(d, x, BLASSUFFIX)
#else
#define BLAS(x) stitch__(s, x, BLASSUFFIX)
#endif
#endif

#ifdef MATLAB_MEX_FILE
typedef ptrdiff_t blas_int;
#elif defined BLAS64
#include <stdint.h>
typedef int64_t blas_int;
#else
typedef int blas_int;
```

```
#endif

#ifdef __cplusplus
}
#endif

#endif

#endif
```

5.48 include/adaptive.h File Reference

```
#include "abip.h"
#include "glbopts.h"
#include <math.h>
```

Functions

- ABIPAdaptWork *ABIP() init_adapt (ABIPWork *w)
- void ABIP() free_adapt (ABIPAdaptWork *a)
- abip_int ABIP() adaptive (ABIPWork *w, abip_int iter)
- char *ABIP() get_adapt_summary (const ABIPInfo *info, ABIPAdaptWork *a)

5.48.1 Function Documentation

5.48.1.1 adaptive()

```
abip_int ABIP() adaptive (
            ABIPWork * w,
            abip_int iter )
```

Definition at line 305 of file adaptive.c.

5.48.1.2 free_adapt()

```
void ABIP() free_adapt (
            ABIPAdaptWork * a )
```

Definition at line 336 of file adaptive.c.

5.48.1.3 get_adapt_summary()

```
char *ABIP() get_adapt_summary (
             const ABIPInfo * info,
             ABIPAdaptWork * a )
```

Definition at line 406 of file adaptive.c.

5.48.1.4 init_adapt()

```
ABIPAdaptWork *ABIP() init_adapt (
             ABIPWork * w )
```

Definition at line 258 of file adaptive.c.

5.49 adaptive.h

Go to the documentation of this file.

```
#ifndef ADAPT_H_GUARD
#define ADAPT_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "abip.h"
#include "glbopts.h"
#include <math.h>

ABIPAdaptWork *ABIP(init_adapt)
(
    ABIPWork *w
);

void ABIP(free_adapt)
(
    ABIPAdaptWork *a
);

abip_int ABIP(adaptive)
(
    ABIPWork *w,
    abip_int iter
);

char *ABIP(get_adapt_summary)
(
    const ABIPInfo *info,
    ABIPAdaptWork *a
);

#ifdef __cplusplus
}
#endif
#endif
```

5.50 include/cs.h File Reference

```
#include "glbopts.h"
```

Functions

- struct ABIP (cs_sparse)
- cs *ABIP() cs_compress (const cs *T)
- cs *ABIP() cs_spalloc (abip_int m, abip_int n, abip_int nnzmax, abip_int values, abip_int triplet)
- cs *ABIP() cs_spfree (cs *A)
- abip_float ABIP() cs_cumsum (abip_int *p, abip_int *c, abip_int n)
- cs *ABIP() cs_transpose (const cs *A, abip_int values)
- abip_int *ABIP() cs_pinv (abip_int const *p, abip_int n)
- cs *ABIP() cs_symperm (const cs *A, const abip_int *pinv, abip_int values)

Variables

- cs

5.50.1 Function Documentation

5.50.1.1 ABIP()

```
struct ABIP (
            cs_sparse )
```

Definition at line 1 of file cs.h.

5.50.1.2 cs_compress()

```
cs *ABIP() cs_compress (
            const cs * T )
```

Definition at line 57 of file cs.c.

5.50.1.3 cs_cumsum()

```
abip_float ABIP() cs_cumsum (
            abip_int * p,
            abip_int * c,
            abip_int n )
```

Definition at line 162 of file cs.c.

5.50.1.4 cs_pinv()

```
abip_int *ABIP() cs_pinv (
            abip_int const * p,
            abip_int n )
```

Definition at line 190 of file cs.c.

5.50.1.5 cs_spalloc()

```
cs *ABIP() cs_spalloc (
            abip_int m,
            abip_int n,
            abip_int nnzmax,
            abip_int values,
            abip_int triplet )
```

Definition at line 117 of file cs.c.

5.50.1.6 cs_spfree()

```
cs *ABIP() cs_spfree (
            cs * A )
```

Definition at line 145 of file cs.c.

5.50.1.7 cs_symperm()

```
cs *ABIP() cs_symperm (
            const cs * A,
            const abip_int * pinv,
            abip_int values )
```

Definition at line 218 of file cs.c.

5.50.1.8 cs_transpose()

```
cs *ABIP() cs_transpose (
            const cs * A,
            abip_int values )
```

Definition at line 33 of file cs.c.

5.50.2 Variable Documentation

5.50.2.1 cs

```
cs
```

Definition at line 19 of file cs.h.

5.51 cs.h

Go to the documentation of this file.

```
00001 #ifndef CS_H_GUARD
00002 #define CS_H_GUARD
00003
00004 #ifdef __cplusplus
00005 extern "C" {
00006 #endif
00007
00008 #include "glbopts.h"
00009
00010 typedef struct ABIP(cs_sparse)
00011 {
00012        abip_int nnzmax;
00013        abip_int m;
00014        abip_int n;
00015        abip_int *p;
00016        abip_int *i;
00017        abip_float *x;
00018        abip_int nnz;
00019 } cs;
00020
00021 cs *ABIP(cs_compress)
00022 (
00023     const cs *T
00024 );
00025
00026 cs *ABIP(cs_spalloc)
00027 (
00028     abip_int m,
00029     abip_int n,
00030     abip_int nnzmax,
00031     abip_int values,
00032     abip_int triplet
00033 );
00034
00035 cs *ABIP(cs_spfree)
00036 (
00037     cs *A
00038 );
00039
00040 abip_float ABIP(cs_cumsum)
00041 (
00042     abip_int *p,
00043     abip_int *c,
00044     abip_int n
00045 );
00046
00047 cs *ABIP(cs_transpose)
00048 (
00049     const cs *A,
00050     abip_int values
00051 );
00052
00053 abip_int *ABIP(cs_pinv)
00054 (
00055     abip_int const *p,
00056     abip_int n
00057 );
00058
00059 cs *ABIP(cs_symperm)
00060 (
00061     const cs *A,
```

```
    const abip_int *pinv,
    abip_int values
);

#ifdef __cplusplus
}
#endif
#endif
```

5.52 include/ctrlc.h File Reference

Macros

- #define abip_start_interrupt_listener()
- #define abip_end_interrupt_listener()
- #define abip_is_interrupted() 0

Typedefs

- typedef int abip_make_iso_compilers_happy

5.52.1 Macro Definition Documentation

5.52.1.1 abip_end_interrupt_listener

```
#define abip_end_interrupt_listener( )
```

Definition at line 36 of file ctrlc.h.

5.52.1.2 abip_is_interrupted

```
#define abip_is_interrupted( ) 0
```

Definition at line 37 of file ctrlc.h.

5.52.1.3 abip_start_interrupt_listener

```
#define abip_start_interrupt_listener( )
```

Definition at line 35 of file ctrlc.h.

5.52.2 Typedef Documentation

5.52.2.1 abip_make_iso_compilers_happy

```
typedef int abip_make_iso_compilers_happy
```

Definition at line 33 of file ctrlc.h.

5.53 ctrlc.h

Go to the documentation of this file.

```
00001 /* Interface for ABIP signal handling. */
00002
00003 #ifndef CTRLC_H_GUARD
00004 #define CTRLC_H_GUARD
00005
00006 #ifdef __cplusplus
00007 extern "C" {
00008 #endif
00009
00010 #if CTRLC > 0
00011
00012 #if defined MATLAB_MEX_FILE
00013
00014 extern int utIsInterruptPending();
00015 extern int utSetInterruptEnabled(int);
00016
00017 #elif(defined _WIN32 || defined _WIN64 || defined _WINDLL)
00018
00019 #include <windows.h>
00020
00021 #else
00022
00023 #include <signal.h>
00024
00025 #endif
00026
00027 void abip_start_interrupt_listener(void);
00028 void abip_end_interrupt_listener(void);
00029 int abip_is_interrupted(void);
00030
00031 #else
00032
00033 typedef int abip_make_iso_compilers_happy;
00034
00035 #define abip_start_interrupt_listener()
00036 #define abip_end_interrupt_listener()
00037 #define abip_is_interrupted() 0
00038
00039 #endif
00040
00041 #ifdef __cplusplus
00042 }
00043 #endif
00044 #endif
```

5.54 include/glbopts.h File Reference

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
```

Macros

- #define ABIP(x) abip_##x
- #define ABIP_VERSION ("2.0.0") /* string literals automatically null-terminated */
- #define ABIP_INFEASIBLE_INACCURATE (-7)
- #define ABIP_UNBOUNDED_INACCURATE (-6)
- #define ABIP_SIGINT (-5)
- #define ABIP_FAILED (-4)
- #define ABIP_INDETERMINATE (-3)
- #define ABIP_INFEASIBLE (-2)
- #define ABIP_UNBOUNDED (-1)
- #define ABIP_UNFINISHED (0)
- #define ABIP_SOLVED (1)
- #define ABIP_SOLVED_INACCURATE (2)
- #define MAX_IPM_ITERS (500)
- #define MAX_ADMM_ITERS (1000000)
- #define EPS (1E-3)
- #define ALPHA (1.8)
- #define CG_RATE (2.0)
- #define NORMALIZE (1)
- #define SCALE (1.0)
- #define SPARSITY_RATIO (0.01)
- #define RHO_Y (1E-3)
- #define ADAPTIVE (1)
- #define EPS_COR (0.2)
- #define EPS_PEN (0.1)
- #define ADAPTIVE_LOOKBACK (20)
- #define VERBOSE (1)
- #define WARM_START (0)
- #define abip_printf printf
- #define _abip_free free
- #define _abip_malloc malloc
- #define _abip_calloc calloc
- #define _abip_realloc realloc
- #define abip_free(x)
- #define abip_malloc(x) _abip_malloc(x)
- #define abip_calloc(x, y) _abip_calloc(x, y)
- #define abip_realloc(x, y) _abip_realloc(x, y)
- #define NAN ((scs_float)0x7ff8000000000000)
- #define INFINITY NAN
- #define ABIP_NULL 0
- #define MAX(a, b) (((a) > (b)) ? (a) : (b))
- #define MIN(a, b) (((a) < (b)) ? (a) : (b))
- #define ABS(x) (((x) < 0) ? -(x) : (x))
- #define POWF pow
- #define SQRTF sqrt
- #define DEBUG_FUNC
- #define RETURN return
- #define EPS_TOL (1E-18)
- #define SAFEDIV_POS(X, Y) ((Y) < EPS_TOL ? ((X) / EPS_TOL) : (X) / (Y))
- #define CONVERGED_INTERVAL (1)
- #define INDETERMINATE_TOL (1e-9)

Typedefs

- typedef int abip_int
- typedef double abip_float

5.54.1 Macro Definition Documentation

5.54.1.1 _abip_calloc

```
#define _abip_calloc calloc
```

Definition at line 75 of file glbopts.h.

5.54.1.2 _abip_free

```
#define _abip_free free
```

Definition at line 73 of file glbopts.h.

5.54.1.3 _abip_malloc

```
#define _abip_malloc malloc
```

Definition at line 74 of file glbopts.h.

5.54.1.4 _abip_realloc

```
#define _abip_realloc realloc
```

Definition at line 76 of file glbopts.h.

5.54.1.5 ABIP

```
#define ABIP(
              x ) abip_##x
```

Definition at line 11 of file glbopts.h.

5.54.1.6 abip_calloc

```
#define abip_calloc(
              x,
              y ) _abip_calloc(x, y)
```

Definition at line 83 of file glbopts.h.

5.54.1.7 ABIP_FAILED

```
#define ABIP_FAILED (-4)
```

Definition at line 25 of file glbopts.h.

5.54.1.8 abip_free

```
#define abip_free(
              x )
```

Value:

```
_abip_free(x);          \
x = ABIP_NULL
```

Definition at line 79 of file glbopts.h.

5.54.1.9 ABIP_INDETERMINATE

```
#define ABIP_INDETERMINATE (-3)
```

Definition at line 26 of file glbopts.h.

5.54.1.10 ABIP_INFEASIBLE

```
#define ABIP_INFEASIBLE (-2)
```

Definition at line 27 of file glbopts.h.

5.54.1.11 ABIP_INFEASIBLE_INACCURATE

```
#define ABIP_INFEASIBLE_INACCURATE (-7)
```

Definition at line 22 of file glbopts.h.

5.54.1.12 abip_malloc

```
#define abip_malloc(
              x ) _abip_malloc(x)
```

Definition at line 82 of file glbopts.h.

5.54.1.13 ABIP_NULL

```
#define ABIP_NULL 0
```

Definition at line 114 of file glbopts.h.

5.54.1.14 abip_printf

```
#define abip_printf printf
```

Definition at line 72 of file glbopts.h.

5.54.1.15 abip_realloc

```
#define abip_realloc(
              x,
              y ) _abip_realloc(x, y)
```

Definition at line 84 of file glbopts.h.

5.54.1.16 ABIP_SIGINT

```
#define ABIP_SIGINT (-5)
```

Definition at line 24 of file glbopts.h.

5.54.1.17 ABIP_SOLVED

```
#define ABIP_SOLVED (1)
```

Definition at line 30 of file glbopts.h.

5.54.1.18 ABIP_SOLVED_INACCURATE

```
#define ABIP_SOLVED_INACCURATE (2)
```

Definition at line 31 of file glbopts.h.

5.54.1.19 ABIP_UNBOUNDED

```
#define ABIP_UNBOUNDED (-1)
```

Definition at line 28 of file glbopts.h.

5.54.1.20 ABIP_UNBOUNDED_INACCURATE

```
#define ABIP_UNBOUNDED_INACCURATE (-6)
```

Definition at line 23 of file glbopts.h.

5.54.1.21 ABIP_UNFINISHED

```
#define ABIP_UNFINISHED (0)
```

Definition at line 29 of file glbopts.h.

5.54.1.22 ABIP_VERSION

```
#define ABIP_VERSION  ("2.0.0") /* string literals automatically null-terminated */
```

Definition at line 19 of file glbopts.h.

5.54.1.23 ABS

```
#define ABS(
            x ) (((x) < 0) ? -(x) : (x))
```

Definition at line 125 of file glbopts.h.

5.54.1.24 ADAPTIVE

```
#define ADAPTIVE (1)
```

Definition at line 42 of file glbopts.h.

5.54.1.25 ADAPTIVE_LOOKBACK

```
#define ADAPTIVE_LOOKBACK (20)
```

Definition at line 45 of file glbopts.h.

5.54.1.26 ALPHA

```
#define ALPHA (1.8)
```

Definition at line 36 of file glbopts.h.

5.54.1.27 CG_RATE

```
#define CG_RATE (2.0)
```

Definition at line 37 of file glbopts.h.

5.54.1.28 CONVERGED_INTERVAL

```
#define CONVERGED_INTERVAL (1)
```

Definition at line 160 of file glbopts.h.

5.54.1.29 DEBUG_FUNC

```
#define DEBUG_FUNC
```

Definition at line 153 of file glbopts.h.

5.54.1.30 EPS

```
#define EPS (1E-3)
```

Definition at line 35 of file glbopts.h.

5.54.1.31 EPS_COR

```
#define EPS_COR (0.2)
```

Definition at line 43 of file glbopts.h.

5.54.1.32 EPS_PEN

```
#define EPS_PEN (0.1)
```

Definition at line 44 of file glbopts.h.

5.54.1.33 EPS_TOL

```
#define EPS_TOL (1E-18)
```

Definition at line 157 of file glbopts.h.

5.54.1.34 INDETERMINATE_TOL

```
#define INDETERMINATE_TOL (1e-9)
```

Definition at line 161 of file glbopts.h.

5.54.1.35 INFINITY

```
#define INFINITY NAN
```

Definition at line 102 of file glbopts.h.

5.54.1.36 MAX

```
#define MAX(
            a,
            b ) (((a) > (b)) ? (a) : (b))
```

Definition at line 117 of file glbopts.h.

5.54.1.37 MAX_ADMM_ITERS

```
#define MAX_ADMM_ITERS (1000000)
```

Definition at line 34 of file glbopts.h.

5.54.1.38 MAX_IPM_ITERS

```
#define MAX_IPM_ITERS (500)
```

Definition at line 33 of file glbopts.h.

5.54.1.39 MIN

```
#define MIN(
            a,
            b ) (((a) < (b)) ? (a) : (b))
```

Definition at line 121 of file glbopts.h.

5.54.1.40 NAN

```
#define NAN ((scs_float)0x7ff8000000000000)
```

Definition at line 99 of file glbopts.h.

5.54.1.41 NORMALIZE

```
#define NORMALIZE (1)
```

Definition at line 38 of file glbopts.h.

5.54.1.42 POWF

```
#define POWF pow
```

Definition at line 132 of file glbopts.h.

5.54.1.43 RETURN

```
#define RETURN return
```

Definition at line 154 of file glbopts.h.

5.54.1.44 RHO_Y

```
#define RHO_Y (1E-3)
```

Definition at line 41 of file glbopts.h.

5.54.1.45 SAFEDIV_POS

```
#define SAFEDIV_POS(
              X,
              Y ) ((Y) < EPS_TOL ? ((X) / EPS_TOL) : (X) / (Y))
```

Definition at line 158 of file glbopts.h.

5.54.1.46 SCALE

```
#define SCALE (1.0)
```

Definition at line 39 of file glbopts.h.

5.54.1.47 SPARSITY_RATIO

```
#define SPARSITY_RATIO (0.01)
```

Definition at line 40 of file glbopts.h.

5.54.1.48 SQRTF

```
#define SQRTF sqrt
```

Definition at line 140 of file glbopts.h.

5.54.1.49 VERBOSE

```
#define VERBOSE (1)
```

Definition at line 46 of file glbopts.h.

5.54.1.50 WARM_START

```
#define WARM_START (0)
```

Definition at line 47 of file glbopts.h.

5.54.2 Typedef Documentation

5.54.2.1 abip_float

```
typedef double abip_float
```

Definition at line 97 of file glbopts.h.

5.54.2.2 abip_int

```
typedef int abip_int
```

Definition at line 93 of file glbopts.h.

5.55 glbopts.h

Go to the documentation of this file.

```
#ifndef GLB_H_GUARD
#define GLB_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include <math.h>

#ifndef ABIP
#define ABIP(x) abip_##x
#endif

//#ifndef ABIP_PARDISO
//#define ABIP_PARDISO
//#endif

/* ABIP VERSION NUMBER ----------------------------------------------    */
#define ABIP_VERSION                                                          \
  ("2.0.0") /* string literals automatically null-terminated */

#define ABIP_INFEASIBLE_INACCURATE (-7)
#define ABIP_UNBOUNDED_INACCURATE (-6)
#define ABIP_SIGINT (-5)
#define ABIP_FAILED (-4)
#define ABIP_INDETERMINATE (-3)
#define ABIP_INFEASIBLE (-2)
#define ABIP_UNBOUNDED (-1)
#define ABIP_UNFINISHED (0)
#define ABIP_SOLVED (1)
#define ABIP_SOLVED_INACCURATE (2)

#define MAX_IPM_ITERS (500)
#define MAX_ADMM_ITERS (1000000)
#define EPS (1E-3)
#define ALPHA (1.8)
#define CG_RATE (2.0)
#define NORMALIZE (1)
#define SCALE (1.0)
#define SPARSITY_RATIO (0.01)
#define RHO_Y (1E-3)
#define ADAPTIVE (1)
#define EPS_COR (0.2)
#define EPS_PEN (0.1)
#define ADAPTIVE_LOOKBACK (20)
#define VERBOSE (1)
#define WARM_START (0)

#ifdef MATLAB_MEX_FILE
#include "mex.h"
#define abip_printf mexPrintf
#define _abip_free mxFree
#define _abip_malloc mxMalloc
#define _abip_calloc mxCalloc
#define _abip_realloc mxRealloc
#elif defined PYTHON
#include <Python.h>
#include <stdlib.h>
#define abip_printf(...)                                                          \
{                                                                                         \
      PyGILState_STATE gilstate = PyGILState_Ensure();        \
      PySys_WriteStdout(__VA_ARGS__);                         \
      PyGILState_Release(gilstate);                                 \
}
#define _abip_free free
#define _abip_malloc malloc
#define _abip_calloc calloc
#define _abip_realloc realloc
#else
#include <stdio.h>
#include <stdlib.h>
#define abip_printf printf
#define _abip_free free
#define _abip_malloc malloc
#define _abip_calloc calloc
#define _abip_realloc realloc
#endif

#define abip_free(x)    \
      _abip_free(x);          \
      x = ABIP_NULL
#define abip_malloc(x) _abip_malloc(x)
```

```
#define abip_calloc(x, y) _abip_calloc(x, y)
#define abip_realloc(x, y) _abip_realloc(x, y)

#ifdef DLONG
#ifdef _WIN64
typedef __int64 abip_int;
#else
typedef long abip_int;
#endif
#else
typedef int abip_int;
#endif

#ifndef SFLOAT
typedef double abip_float;
#ifndef NAN
#define NAN ((scs_float)0x7ff8000000000000)
#endif
#ifndef INFINITY
#define INFINITY NAN
#endif
#else
typedef float abip_float;
#ifndef NAN
#define NAN ((float)0x7fc00000)
#endif
#ifndef INFINITY
#define INFINITY NAN
#endif
#endif

#define ABIP_NULL 0

#ifndef MAX
#define MAX(a, b) (((a) > (b)) ? (a) : (b))
#endif

#ifndef MIN
#define MIN(a, b) (((a) < (b)) ? (a) : (b))
#endif

#ifndef ABS
#define ABS(x) (((x) < 0) ? -(x) : (x))
#endif

#ifndef POWF
#ifdef SFLOAT
#define POWF powf
#else
#define POWF pow
#endif
#endif

#ifndef SQRTF
#ifdef SFLOAT
#define SQRTF sqrtf
#else
#define SQRTF sqrt
#endif
#endif

#if EXTRA_VERBOSE > 1
#if (defined _WIN32 || defined _WIN64 || defined _WINDLL)
#define __func__ __FUNCTION__
#endif
#define DEBUG_FUNC abip_printf("IN function: %s, time: %4f ms, file: %s, line: %i\n", __func__,
 ABIP(tocq)(&global_timer), __FILE__, __LINE__);
#define RETURN
      abip_printf("EXIT function: %s, time: %4f ms, file: %s, line: %i\n", __func__,
 ABIP(tocq)(&global_timer), __FILE__, __LINE__);    \
      return
#else
#define DEBUG_FUNC
#define RETURN return
#endif

#define EPS_TOL (1E-18)
#define SAFEDIV_POS(X, Y) ((Y) < EPS_TOL ? ((X) / EPS_TOL) : (X) / (Y))

#define CONVERGED_INTERVAL (1)
#define INDETERMINATE_TOL (1e-9)

#ifdef __cplusplus
}
#endif
#endif
```

5.56 include/linalg.h File Reference

```
#include "abip.h"
#include <math.h>
```

Functions

- void ABIP() set_as_scaled_array (abip_float *x, const abip_float *a, const abip_float b, abip_int len)

  *compute x = b*a*
- void ABIP() set_as_sqrt (abip_float *x, const abip_float *v, abip_int len)

  compute x = sqrt(v)
- void ABIP() set_as_sq (abip_float *x, const abip_float *v, abip_int len)

  compute x = v.^2
- void ABIP() scale_array (abip_float *a, const abip_float b, abip_int len)

  *compute a *= b*
- abip_float ABIP() dot (const abip_float *x, const abip_float *y, abip_int len)

  *compute x'*y*
- abip_float ABIP() norm_sq (const abip_float *v, abip_int len)

  compute||v||_2^2
- abip_float ABIP() norm (const abip_float *v, abip_int len)

  compute ||v||_2
- abip_float ABIP() norm_inf (const abip_float *a, abip_int len)

  compute the infinity norm
- abip_float ABIP() norm_one (const abip_float *v, abip_int len)

  compute L1 norm
- abip_float ABIP() norm_one_sqrt (const abip_float *v, abip_int len)

  compute square root L1 norm
- abip_float ABIP() norm_inf_sqrt (const abip_float *v, abip_int len)

  compute square root infinity norm
- abip_float ABIP() min_abs_sqrt (const abip_float *a, abip_int len, abip_float ref)

  compute square root of the minimal absolute value
- void ABIP() add_array (abip_float *a, const abip_float b, abip_int len)

  compute a .+= b
- void ABIP() add_scaled_array (abip_float *a, const abip_float *b, abip_int n, const abip_float sc)

  *compute a += sc*b*
- abip_float ABIP() norm_diff (const abip_float *a, const abip_float *b, abip_int len)

  compute ||a-b||_2^2
- abip_float ABIP() norm_inf_diff (const abip_float *a, const abip_float *b, abip_int len)

  compute max(|a-b|)

5.56.1 Function Documentation

5.56.1.1 add_array()

```
void ABIP() add_array (
             abip_float * a,
             const abip_float b,
             abip_int len )
```

compute a .+= b

Definition at line 218 of file linalg.c.

5.56.1.2 add_scaled_array()

```
void ABIP() add_scaled_array (
             abip_float * a,
             const abip_float * b,
             abip_int n,
             const abip_float sc )
```

compute a += sc∗b

Definition at line 235 of file linalg.c.

5.56.1.3 dot()

```
abip_float ABIP() dot (
             const abip_float * x,
             const abip_float * y,
             abip_int len )
```

compute x'∗y

Definition at line 78 of file linalg.c.

5.56.1.4 min_abs_sqrt()

```
abip_float ABIP() min_abs_sqrt (
             const abip_float * a,
             abip_int len,
             abip_float ref )
```

compute square root of the minimal absolute value

Definition at line 126 of file linalg.c.

5.56.1.5 norm()

```
abip_float ABIP() norm (
            const abip_float * v,
            abip_int len )
```

compute ||v||_2

Definition at line 115 of file linalg.c.

5.56.1.6 norm_diff()

```
abip_float ABIP() norm_diff (
            const abip_float * a,
            const abip_float * b,
            abip_int len )
```

compute ||a-b||_2^2

Definition at line 253 of file linalg.c.

5.56.1.7 norm_inf()

```
abip_float ABIP() norm_inf (
            const abip_float * a,
            abip_int len )
```

compute the infinity norm

Definition at line 182 of file linalg.c.

5.56.1.8 norm_inf_diff()

```
abip_float ABIP() norm_inf_diff (
            const abip_float * a,
            const abip_float * b,
            abip_int len )
```

compute max(|a-b|)

Definition at line 274 of file linalg.c.

5.56.1.9 norm_inf_sqrt()

```
abip_float ABIP() norm_inf_sqrt (
            const abip_float * v,
            abip_int len )
```

compute square root infinity norm

Definition at line 205 of file linalg.c.

5.56.1.10 norm_one()

```
abip_float ABIP() norm_one (
            const abip_float * v,
            abip_int len )
```

compute L1 norm

Definition at line 149 of file linalg.c.

5.56.1.11 norm_one_sqrt()

```
abip_float ABIP() norm_one_sqrt (
            const abip_float * v,
            abip_int len )
```

compute square root L1 norm

Definition at line 167 of file linalg.c.

5.56.1.12 norm_sq()

```
abip_float ABIP() norm_sq (
            const abip_float * v,
            abip_int len )
```

compute||v||_2^2

Definition at line 97 of file linalg.c.

5.56.1.13 scale_array()

```
void ABIP() scale_array (
            abip_float * a,
            const abip_float b,
            abip_int len )
```

compute a *= b

Definition at line 61 of file linalg.c.

5.56.1.14 set_as_scaled_array()

```
void ABIP() set_as_scaled_array (
            abip_float * x,
            const abip_float * a,
            const abip_float b,
            abip_int len )
```

compute x = b*a

Definition at line 9 of file linalg.c.

5.56.1.15 set_as_sq()

```
void ABIP() set_as_sq (
            abip_float * x,
            const abip_float * v,
            abip_int len )
```

compute x = v.^2

Definition at line 44 of file linalg.c.

5.56.1.16 set_as_sqrt()

```
void ABIP() set_as_sqrt (
            abip_float * x,
            const abip_float * v,
            abip_int len )
```

compute x = sqrt(v)

Definition at line 27 of file linalg.c.

5.57 linalg.h

Go to the documentation of this file.

```
#ifndef LINALG_H_GUARD
#define LINALG_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "abip.h"
#include <math.h>

void ABIP(set_as_scaled_array)
(
    abip_float *x,
    const abip_float *a,
    const abip_float b,
    abip_int len
);

void ABIP(set_as_sqrt)
(
    abip_float *x,
    const abip_float *v,
    abip_int len
);

void ABIP(set_as_sq)
(
    abip_float *x,
    const abip_float *v,
    abip_int len
);

void ABIP(scale_array)
(
    abip_float *a,
    const abip_float b,
    abip_int len
);


abip_float ABIP(dot)
(
    const abip_float *x,
    const abip_float *y,
    abip_int len
);

abip_float ABIP(norm_sq)
(
    const abip_float *v,
    abip_int len
);

abip_float ABIP(norm)
(
    const abip_float *v,
    abip_int len
);

abip_float ABIP(norm_inf)
(
    const abip_float *a,
    abip_int len
);

abip_float ABIP(norm_one)
(
      const abip_float *v,
      abip_int len
);

abip_float ABIP(norm_one_sqrt)
(
      const abip_float *v,
      abip_int len
);

abip_float ABIP(norm_inf_sqrt)
(
      const abip_float *v,
      abip_int len
);
```

```

abip_float ABIP(min_abs_sqrt)
(
      const abip_float *a,
      abip_int len,
      abip_float ref
);


void ABIP(add_array)
(
    abip_float *a,
    const abip_float b,
    abip_int len
);

void ABIP(add_scaled_array)
(
    abip_float *a,
    const abip_float *b,
    abip_int n,
    const abip_float sc
);

abip_float ABIP(norm_diff)
(
    const abip_float *a,
    const abip_float *b,
    abip_int len
);

abip_float ABIP(norm_inf_diff)
(
    const abip_float *a,
    const abip_float *b,
    abip_int len
);

#ifdef __cplusplus
}
#endif
#endif
```

5.58 include/linsys.h File Reference

```
#include "abip.h"
```

Functions

- ABIPLinSysWork *ABIP() init_lin_sys_work (const ABIPMatrix *A, const ABIPSettings *stgs)
- abip_int ABIP() solve_lin_sys (const ABIPMatrix *A, const ABIPSettings *stgs, ABIPLinSysWork *p, abip_float *b, const abip_float *s, abip_int iter)
- void ABIP() free_lin_sys_work (ABIPLinSysWork *p)
- void ABIP() free_lin_sys_work_pds (ABIPLinSysWork *p, ABIPMatrix *A)
- void ABIP() accum_by_Atrans (const ABIPMatrix *A, ABIPLinSysWork *p, const abip_float *x, abip_float *y)
- void ABIP() accum_by_A (const ABIPMatrix *A, ABIPLinSysWork *p, const abip_float *x, abip_float *y)
- abip_int ABIP() validate_lin_sys (const ABIPMatrix *A)

  validate the linear system
- char *ABIP() get_lin_sys_method (const ABIPMatrix *A, const ABIPSettings *stgs)
- char *ABIP() get_lin_sys_summary (ABIPLinSysWork *p, const ABIPInfo *info)
- void ABIP() normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, ABIPScaling *scal)
- void ABIP() un_normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, const ABIPScaling *scal)
- void ABIP() free_A_matrix (ABIPMatrix *A)

  set the memory of matrix A free
- abip_int ABIP() copy_A_matrix (ABIPMatrix **dstp, const ABIPMatrix *src)

  copy matrix A

5.58.1 Function Documentation

5.58.1.1 accum_by_A()

```
void ABIP() accum_by_A (
            const ABIPMatrix * A,
            ABIPLinSysWork * p,
            const abip_float * x,
            abip_float * y )
```

Definition at line 205 of file direct.c.

5.58.1.2 accum_by_Atrans()

```
void ABIP() accum_by_Atrans (
            const ABIPMatrix * A,
            ABIPLinSysWork * p,
            const abip_float * x,
            abip_float * y )
```

Definition at line 200 of file direct.c.

5.58.1.3 copy_A_matrix()

```
abip_int ABIP() copy_A_matrix (
            ABIPMatrix ** dstp,
            const ABIPMatrix * src )
```

copy matrix A

Definition at line 10 of file common.c.

5.58.1.4 free_A_matrix()

```
void ABIP() free_A_matrix (
            ABIPMatrix * A )
```

set the memory of matrix A free

Definition at line 99 of file common.c.

5.58.1.5 free_lin_sys_work()

```
void ABIP() free_lin_sys_work (
            ABIPLinSysWork * p )
```

Definition at line 28 of file direct.c.

5.58.1.6 free_lin_sys_work_pds()

```
void ABIP() free_lin_sys_work_pds (
            ABIPLinSysWork * p,
            ABIPMatrix * A )
```

5.58.1.7 get_lin_sys_method()

```
char *ABIP() get_lin_sys_method (
            const ABIPMatrix * A,
            const ABIPSettings * stgs )
```

Definition at line 5 of file direct.c.

5.58.1.8 get_lin_sys_summary()

```
char *ABIP() get_lin_sys_summary (
            ABIPLinSysWork * p,
            const ABIPInfo * info )
```

Definition at line 15 of file direct.c.

5.58.1.9 init_lin_sys_work()

```
ABIPLinSysWork *ABIP() init_lin_sys_work (
            const ABIPMatrix * A,
            const ABIPSettings * stgs )
```

Definition at line 273 of file direct.c.

5.58.1.10 normalize_A()

```
void ABIP() normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPScaling * scal )
```

Definition at line 210 of file direct.c.

5.58.1.11 solve_lin_sys()

```
abip_int ABIP() solve_lin_sys (
            const ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPLinSysWork * p,
            abip_float * b,
            const abip_float * s,
            abip_int iter )
```

Definition at line 305 of file direct.c.

5.58.1.12 un_normalize_A()

```
void ABIP() un_normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            const ABIPScaling * scal )
```

Definition at line 214 of file direct.c.

5.58.1.13 validate_lin_sys()

```
abip_int ABIP() validate_lin_sys (
            const ABIPMatrix * A )
```

validate the linear system

Definition at line 45 of file common.c.

5.59 linsys.h

Go to the documentation of this file.

```
#ifndef LINSYS_H_GUARD
#define LINSYS_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "abip.h"

ABIPLinSysWork *ABIP(init_lin_sys_work)(const ABIPMatrix *A, const ABIPSettings *stgs);

abip_int ABIP(solve_lin_sys)
(
    const ABIPMatrix *A,
    const ABIPSettings *stgs,
    ABIPLinSysWork *p,
    abip_float *b,
    const abip_float *s,
    abip_int iter
);

void ABIP(free_lin_sys_work)
(
    ABIPLinSysWork *p
);

void ABIP(free_lin_sys_work_pds)
(
    ABIPLinSysWork *p,
    ABIPMatrix *A
);

/* forms y += A'*x */
void ABIP(accum_by_Atrans)
(
    const ABIPMatrix *A,
    ABIPLinSysWork *p,
    const abip_float *x,
    abip_float *y
);

/* forms y += A*x */
void ABIP(accum_by_A)
(
    const ABIPMatrix *A,
    ABIPLinSysWork *p,
    const abip_float *x,
    abip_float *y
);

abip_int ABIP(validate_lin_sys)
(
    const ABIPMatrix *A
);

char *ABIP(get_lin_sys_method)
(
    const ABIPMatrix *A,
    const ABIPSettings *stgs
);

char *ABIP(get_lin_sys_summary)
(
    ABIPLinSysWork *p,
    const ABIPInfo *info
);

void ABIP(normalize_A)
(
    ABIPMatrix *A,
    const ABIPSettings *stgs,
    ABIPScaling *scal
);

void ABIP(un_normalize_A)
(
    ABIPMatrix *A,
    const ABIPSettings *stgs,
    const ABIPScaling *scal
);

void ABIP(free_A_matrix)
```

```
(
    ABIPMatrix *A
);

abip_int ABIP(copy_A_matrix)
(
    ABIPMatrix **dstp,
    const ABIPMatrix *src
);

#ifdef __cplusplus
}
#endif

#endif
```

5.60 include/normalize.h File Reference

```
#include "abip.h"
```

Functions

- void ABIP() normalize_b_c (ABIPWork *w)

  normalize b and c
- void ABIP() calc_scaled_resids (ABIPWork *w, ABIPResiduals *r)

  calculate the scaled residuals
- void ABIP() normalize_warm_start (ABIPWork *w)

  normalize the warm start solution
- void ABIP() un_normalize_sol (ABIPWork *w, ABIPSolution *sol)

  recover the optimal solution

5.60.1 Function Documentation

5.60.1.1 calc_scaled_resids()

```
void ABIP() calc_scaled_resids (
            ABIPWork * w,
            ABIPResiduals * r )
```

calculate the scaled residuals

Definition at line 44 of file normalize.c.

5.60.1.2 normalize_b_c()

```
void ABIP() normalize_b_c (
            ABIPWork * w )
```

normalize b and c

Definition at line 11 of file normalize.c.

5.60.1.3 normalize_warm_start()

```
void ABIP() normalize_warm_start (
             ABIPWork * w )
```

normalize the warm start solution

Definition at line 100 of file normalize.c.

5.60.1.4 un_normalize_sol()

```
void ABIP() un_normalize_sol (
             ABIPWork * w,
             ABIPSolution * sol )
```

recover the optimal solution

Definition at line 133 of file normalize.c.

5.61 normalize.h

Go to the documentation of this file.

```
#ifndef NORMALIZE_H_GUARD
#define NORMALIZE_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "abip.h"

void ABIP(normalize_b_c)
(
    ABIPWork *w
);

void ABIP(calc_scaled_resids)
(
    ABIPWork *w,
    ABIPResiduals *r
);

void ABIP(normalize_warm_start)
(
    ABIPWork *w
);

void ABIP(un_normalize_sol)
(
    ABIPWork *w,
    ABIPSolution *sol
);

#ifdef __cplusplus
}
#endif
#endif
```

5.62 include/util.h File Reference

```
#include "abip.h"
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
```

Functions

- struct ABIP (timer)
- void ABIP() tic (ABIP(timer) ∗t)
- abip_float ABIP() toc (ABIP(timer) ∗t)

    define toc function
- abip_float ABIP() str_toc (char ∗str, ABIP(timer) ∗t)

    store time consumed
- abip_float ABIP() tocq (ABIP(timer) ∗t)
- void ABIP() print_data (const ABIPData ∗d)

    print some parameters
- void ABIP() print_work (const ABIPWork ∗w)

    print the iterates
- void ABIP() print_array (const abip_float ∗arr, abip_int n, const char ∗name)

    print array
- void ABIP() set_default_settings (ABIPData ∗d)

    set default setting
- void ABIP() free_sol (ABIPSolution ∗sol)

    set the memory of solution free
- void ABIP() free_data (ABIPData ∗d)

    set the memory of problem data free

5.62.1 Function Documentation

5.62.1.1 ABIP()

```
ABIP (
          timer )
```

Definition at line 1 of file util.h.

5.62.1.2 free_data()

```
void ABIP() free_data (
          ABIPData * d )
```

set the memory of problem data free

Definition at line 226 of file util.c.

5.62.1.3 free_sol()

```
void ABIP() free_sol (
            ABIPSolution * sol )
```

set the memory of solution free

Definition at line 259 of file util.c.

5.62.1.4 print_array()

```
void ABIP() print_array (
            const abip_float * arr,
            abip_int n,
            const char * name )
```

print array

Definition at line 191 of file util.c.

5.62.1.5 print_data()

```
void ABIP() print_data (
            const ABIPData * d )
```

print some parameters

Definition at line 162 of file util.c.

5.62.1.6 print_work()

```
void ABIP() print_work (
            const ABIPWork * w )
```

print the iterates

Definition at line 133 of file util.c.

5.62.1.7 set_default_settings()

```
void ABIP() set_default_settings (
            ABIPData * d )
```

set default setting

Definition at line 288 of file util.c.

5.62.1.8 str_toc()

```
abip_float ABIP() str_toc (
            char * str,
            ABIP(timer) * t )
```

store time consumed

Definition at line 120 of file util.c.

5.62.1.9 tic()

```
void ABIP() tic (
            ABIP(timer) * t )
```

Definition at line 73 of file util.c.

5.62.1.10 toc()

```
abip_float ABIP() toc (
            ABIP(timer) * t )
```

define toc function

Definition at line 108 of file util.c.

5.62.1.11 tocq()

```
abip_float ABIP() tocq (
            ABIP(timer) * t )
```

Definition at line 81 of file util.c.

5.63 util.h

Go to the documentation of this file.

```
#ifndef UTIL_H_GUARD
#define UTIL_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "abip.h"
#include <stdlib.h>
#include <stdio.h>

#if (defined NOTIMER)
typedef void *ABIP(timer);

#elif(defined _WIN32 || defined _WIN64 || defined _WINDLL)

#include <windows.h>
typedef struct ABIP(timer)
{
      LARGE_INTEGER tic;
      LARGE_INTEGER toc;
      LARGE_INTEGER freq;
} ABIP(timer);

#elif(defined __APPLE__)

#include <mach/mach_time.h>
typedef struct ABIP(timer)
{
      uint64_t tic;
      uint64_t toc;
      mach_timebase_info_data_t tinfo;
} ABIP(timer);

#else

#include <time.h>
typedef struct ABIP(timer)
{
      struct timespec tic;
      struct timespec toc;
} ABIP(timer);

#endif

#if EXTRA_VERBOSE > 1
extern ABIP(timer) global_timer;
#endif

void ABIP(tic)(ABIP(timer) *t);
abip_float ABIP(toc)(ABIP(timer) *t);
abip_float ABIP(str_toc)(char *str, ABIP(timer) *t);
abip_float ABIP(tocq)(ABIP(timer) *t);

void ABIP(print_data)(const ABIPData *d);
void ABIP(print_work)(const ABIPWork *w);
void ABIP(print_array)(const abip_float *arr, abip_int n, const char *name);
void ABIP(set_default_settings)(ABIPData *d);
void ABIP(free_sol)(ABIPSolution *sol);
void ABIP(free_data)(ABIPData *d);

#ifdef __cplusplus
}
#endif
#endif
```

5.64 interface/abip_direct.m File Reference

Functions

- s t A in R (m ∗n) and b \in R^m. % % this uses the direct linear equation solver version of ABIP. % % data must consist of data.A

Variables

- function [x, y, s, info]
- s t Ax = b
- s t x
- s t A in data b
- s t A in data data c
- s t A in data data where A

5.64.1 Function Documentation

5.64.1.1 R()

```
s t A in R (
             m * n )
```

5.64.2 Variable Documentation

5.64.2.1 A

```
s t A in data data where A
```

Definition at line 15 of file abip_direct.m.

5.64.2.2 Ax

```
s t Ax = b
```

Definition at line 9 of file abip_direct.m.

5.64.2.3 b

```
s t A in data data where b
```

Definition at line 15 of file abip_direct.m.

5.64.2.4 c

```
abip_version c
```

Definition at line 15 of file abip_direct.m.

5.64.2.5 function

```
function[x, y, s, info]
```

Initial value:

```
= abip_direct(data, params)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ADMM-Based Interior-Point Method for solving linear programs (abip_direct):
%
% This implements a LP solver using sparse LDL^T factorization. It solves:
%
% min. c'x
```

Definition at line 1 of file abip_direct.m.

5.64.2.6 x

```
s t x
```

Initial value:

```
=0.
%
% where x \in R^n
```

Definition at line 9 of file abip_direct.m.

5.65 abip_direct.m

Go to the documentation of this file.

```
00001 function [x, y, s, info] = abip_direct(data, params)
00002
00003 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
00004 %% ADMM-Based Interior-Point Method for solving linear programs (abip_direct):
00005 %
00006 % This implements a LP solver using sparse LDL^T factorization. It solves:
00007 %
00008 % min. c'x,
00009 % s.t. Ax = b, x>=0.
00010 %
00011 % where x \in R^n, A \in R^(m*n) and b \in R^m.
00012 %
00013 % this uses the direct linear equation solver version of ABIP.
00014 %
00015 % data must consist of data.A, data.b, data.c, where A,b,c used as above.
00016 %
00017 % Optional fields in the params struct are:
00018 %   max_ipm_iters :     maximum number of IPM iterations.
00019 %   max_admm_iters :    maximum number of ADMM iterations.
00020 %   eps :               quitting tolerance.
00021 %   sigma :             aggressiveness measurement of the IPM framework.
00022 %   alpha :             over-relaxation parameter, between (0,2), alpha=1 is unrelaxed.
00023 %   normalize :         heuristic nomarlization procedure, between 0 and 1, off or on.
00024 %   scale :             heuristic rescale procedure, only used if normalize=1.
00025 %   adaptive :          heuristic barzilai-borwein spectral procedure.
00026 %   verbose :           verbosity level (0 or 1)
00027 %
00028 % to warm-start the solver add guesses for (x, y, s) to the data struct
00029 %
00030 error ('abip_direct mexFunction not found') ;
```

5.66 interface/abip_indirect.m File Reference

Functions

- s t A in R (m ∗n) and b \in R^m. % % this uses the indirect linear equation solver version of ABIP. % % data must consist of data.A

Variables

- function [x, y, s, info]
- s t Ax = b
- s t x
- s t A in data b
- s t A in data data c
- s t A in data data where A

5.66.1 Function Documentation

5.66.1.1 R()

```
s t A in R (
            m * n )
```

5.66.2 Variable Documentation

5.66.2.1 A

```
s t A in data data where A
```

Definition at line 15 of file abip_indirect.m.

5.66.2.2 Ax

```
s t Ax = b
```

Definition at line 9 of file abip_indirect.m.

5.66.2.3 b

```
s t A in data data where b
```

Definition at line 15 of file abip_indirect.m.

5.66.2.4 c

```
s t A in data data c
```

Definition at line 15 of file abip_indirect.m.

5.66.2.5 function

```
function[x, y, s, info]
```

Initial value:

```
= abip_indirect(data, params)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ADMM-Based Interior-Point Method for solving linear programs (abip_indirect):
%
% This implements a LP solver using sparse LDL^T factorization. It solves:
%
% min. c'x
```

Definition at line 1 of file abip_indirect.m.

5.66.2.6 x

```
s t x
```

Initial value:

```
=0.
%
% where x \in R^n
```

Definition at line 9 of file abip_indirect.m.

5.67 abip_indirect.m

Go to the documentation of this file.

```
function [x, y, s, info] = abip_indirect(data, params)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ADMM-Based Interior-Point Method for solving linear programs (abip_indirect):
%
% This implements a LP solver using sparse LDL^T factorization. It solves:
%
% min. c'x,
% s.t. Ax = b, x>=0.
%
% where x \in R^n, A \in R^(m*n) and b \in R^m.
%
% this uses the indirect linear equation solver version of ABIP.
%
% data must consist of data.A, data.b, data.c, where A,b,c used as above.
%
% Optional fields in the params struct are:
%   max_ipm_iters :     maximum number of IPM iterations.
%   max_admm_iters :    maximum number of ADMM iterations.
%   eps :               quitting tolerance.
%   sigma :             aggressiveness measurement of the IPM framework.
%   alpha :             over-relaxation parameter, between (0,2), alpha=1 is unrelaxed.
%   normalize :         heuristic nomarlization procedure, between 0 and 1, off or on.
%   scale :             heuristic rescale procedure, only used if normalize=1.
%   adaptive :          heuristic barzilai-borwein spectral procedure.
%   verbose :           verbosity level (0 or 1)
%
% to warm-start the solver add guesses for (x, y, s) to the data struct
%
error ('abip_indirect mexFunction not found') ;
```

5.68 linsys/abip_pardiso.c File Reference

```
#include "abip.h"
#include "cs.h"
#include "amd.h"
#include "ldl.h"
#include "direct.h"
#include "abip_pardiso.h"
```

Functions

- abip_int pardisoFactorize (ABIPLinSysWork *p, cs *A)

  factorize matrix by pardiso
- void pardisoFree (ABIPLinSysWork *p, ABIPMatrix *A)

  Free the internal structure of pardiso.
- void pardisoSolve (ABIPLinSysWork *p, ABIPMatrix *A, abip_float *b)

  Solve the linear system $S * X = B$ *using Pardiso.*

5.68.1 Function Documentation

5.68.1.1 pardisoFactorize()

```
abip_int pardisoFactorize (
            ABIPLinSysWork * p,
            cs * A )
```

factorize matrix by pardiso

Definition at line 23 of file abip_pardiso.c.

5.68.1.2 pardisoFree()

```
void pardisoFree (
            ABIPLinSysWork * p,
            ABIPMatrix * A )
```

Free the internal structure of pardiso.

Definition at line 51 of file abip_pardiso.c.

5.68.1.3 pardisoSolve()

```
void pardisoSolve (
            ABIPLinSysWork * p,
            ABIPMatrix * A,
            abip_float * b )
```

Solve the linear system S ∗ X = B using Pardiso.

Definition at line 66 of file abip_pardiso.c.

5.69 abip_pardiso.c

Go to the documentation of this file.

```
00001 #include "abip.h"
00002 #include "cs.h"
00003 #include "amd.h"
00004 #include "ldl.h"
00005 #include "direct.h"
00006 #include "abip_pardiso.h"
00010 static void printMat( cs *A ) {
00011     abip_int *Ap = A->p, *Ai = A->i; abip_float *Ax = A->x;
00012     abip_printf("Matrix dimension: %d by %d \n", A->m, A->n);
00013     abip_printf("%8s %8s %8s \n", "i", "j", "val");
00014     for (abip_int i = 0, j; i < A->m; ++i) {
00015         for (j = Ap[i]; j < Ap[i + 1]; ++j) {
00016             abip_printf("%8d %8d %8.2e \n", Ai[j], i, Ax[j]);
00017         }
00018     }
00019 }
00023 extern abip_int pardisoFactorize( ABIPLinSysWork *p, cs *A ) {
00024     /* Factorize the spsMat matrix */
00025     abip_int phase = PARDISO_SYM_FAC, error = PARDISO_OK;
00026     pardiso(p->pardiso_work, &maxfct, &mnum, &mtype, &phase, &A->n,
00027             A->x, A->p, A->i, p->P, &idummy, PARDISO_PARAMS_LDL,
```

```
            &msglvl, NULL, NULL, &error);

    if (!p->i) {
        p->i = (abip_int *) abip_calloc((A->m + 1), sizeof(abip_int));
    }

    if (!p->j) {
        p->j = (abip_int *) abip_calloc(A->p[A->m], sizeof(abip_int));
    }

    memcpy(p->i, A->p, sizeof(abip_int) * (A->m + 1));
    memcpy(p->j, A->i, sizeof(abip_int) * A->p[A->m]);

    if (error) {
        abip_printf("[Pardiso Error]: Matrix factorization failed."
               " Error code: %d \n", error);
    }

    return error;
}
extern void pardisoFree( ABIPLinSysWork *p, ABIPMatrix *A ) {
    abip_int phase = PARDISO_FREE, error = PARDISO_OK, n = A->m + A->n, one = 1;
    pardiso(p->pardiso_work, &maxfct, &mnum, &mtype, &phase, &n,
            NULL, p->i, p->j, &idummy, &one,
            PARDISO_PARAMS_LDL, &msglvl, &ddummy, &ddummy, &error);
    abip_free(p->i); abip_free(p->j);
    if (error) {
        abip_printf("[Pardiso Error]: Pardiso free failed."
               " Error code %d \n", error);
    }
}

extern void pardisoSolve( ABIPLinSysWork *p, ABIPMatrix *A, abip_float *b ) {

    abip_int phase = PARDISO_SOLVE, error = PARDISO_OK, nrhs = 1, n = A->m + A->n;
    pardiso(p->pardiso_work, &maxfct, &mnum, &mtype, &phase, &n,
            NULL, p->i, p->j, &idummy, &nrhs, PARDISO_PARAMS_LDL,
            &msglvl, b, p->D, &error);
    if (error) {
        abip_printf("[Pardiso Error]: Pardiso solve failed."
               " Error code %d \n", error);
    }
}

```

5.70 linsys/abip_pardiso.h File Reference

```
#include <stdio.h>
#include "glbopts.h"
```

Macros

- #define PARDISO_OK (0)
- #define PARDISOINDEX (64)
- #define PARDISO_SYM (11)
- #define PARDISO_FAC (22)
- #define PARDISO_SYM_FAC (12)
- #define PARDISO_SOLVE (33)
- #define PARDISO_FORWARD (331)
- #define PARDISO_BACKWARD (333)
- #define PARDISO_FREE (-1)
- #define SYMBOLIC 3
- #define PIVOTING 2
- #define FACTORIZE 0

Functions

- void pardisoinit (void ∗, const abip_int ∗, abip_int ∗)
- void pardiso (void ∗, abip_int ∗, abip_int ∗, abip_int ∗, abip_int ∗, abip_int ∗, double ∗, abip_int ∗, abip_int ∗, abip_int ∗, abip_int ∗, abip_int ∗, abip_int ∗, double ∗, double ∗, abip_int ∗)
- abip_int pardisoFactorize (ABIPLinSysWork ∗p, cs ∗A)

  factorize matrix by pardiso
- void pardisoFree (ABIPLinSysWork ∗p, ABIPMatrix ∗A)

  Free the internal structure of pardiso.
- void pardisoSolve (ABIPLinSysWork ∗p, ABIPMatrix ∗A, abip_float ∗b)

  Solve the linear system $S * X = B$ using Pardiso.

5.70.1 Macro Definition Documentation

5.70.1.1 FACTORIZE

```
#define FACTORIZE 0
```

Definition at line 30 of file abip_pardiso.h.

5.70.1.2 PARDISO_BACKWARD

```
#define PARDISO_BACKWARD (333)
```

Definition at line 16 of file abip_pardiso.h.

5.70.1.3 PARDISO_FAC

```
#define PARDISO_FAC ( 22)
```

Definition at line 12 of file abip_pardiso.h.

5.70.1.4 PARDISO_FORWARD

```
#define PARDISO_FORWARD (331)
```

Definition at line 15 of file abip_pardiso.h.

5.70.1.5 PARDISO_FREE

```
#define PARDISO_FREE ( -1)
```

Definition at line 17 of file abip_pardiso.h.

5.70.1.6 PARDISO_OK

```
#define PARDISO_OK ( 0)
```

Definition at line 9 of file abip_pardiso.h.

5.70.1.7 PARDISO_SOLVE

```
#define PARDISO_SOLVE ( 33)
```

Definition at line 14 of file abip_pardiso.h.

5.70.1.8 PARDISO_SYM

```
#define PARDISO_SYM ( 11)
```

Definition at line 11 of file abip_pardiso.h.

5.70.1.9 PARDISO_SYM_FAC

```
#define PARDISO_SYM_FAC ( 12)
```

Definition at line 13 of file abip_pardiso.h.

5.70.1.10 PARDISOINDEX

```
#define PARDISOINDEX ( 64)
```

Definition at line 10 of file abip_pardiso.h.

5.70.1.11 PIVOTING

```
#define PIVOTING 2
```

Definition at line 29 of file abip_pardiso.h.

5.70.1.12 SYMBOLIC

```
#define SYMBOLIC 3
```

Definition at line 28 of file abip_pardiso.h.

5.70.2 Function Documentation

5.70.2.1 pardiso()

```
void pardiso (
            void * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            double * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            abip_int * ,
            double * ,
            double * ,
            abip_int * )
```

5.70.2.2 pardisoFactorize()

```
abip_int pardisoFactorize (
            ABIPLinSysWork * p,
            cs * A )
```

factorize matrix by pardiso

Definition at line 23 of file abip_pardiso.c.

5.70.2.3 pardisoFree()

```
void pardisoFree (
            ABIPLinSysWork * p,
            ABIPMatrix * A )
```

Free the internal structure of pardiso.

Definition at line 51 of file abip_pardiso.c.

5.70.2.4 pardisoinit()

```
void pardisoinit (
            void * ,
            const abip_int * ,
            abip_int * )
```

5.70.2.5 pardisoSolve()

```
void pardisoSolve (
            ABIPLinSysWork * p,
            ABIPMatrix * A,
            abip_float * b )
```

Solve the linear system S ∗ X = B using Pardiso.

Definition at line 66 of file abip_pardiso.c.

5.71 abip_pardiso.h

Go to the documentation of this file.

```
#ifndef abip_pardiso_h
#define abip_pardiso_h

/* Implement the pardiso solver interface for IPM */

#include <stdio.h>
#include "glbopts.h"

#define PARDISO_OK         (  0)
#define PARDISOINDEX       ( 64)      // Pardiso working array length
#define PARDISO_SYM        ( 11)      // Pardiso symbolic analysis
#define PARDISO_FAC        ( 22)      // Pardiso numerical factorization
#define PARDISO_SYM_FAC    ( 12)      // Symbolic analysis and factorization
#define PARDISO_SOLVE      ( 33)      // Solve linear system
#define PARDISO_FORWARD    (331)      // Pardiso forward solve
#define PARDISO_BACKWARD   (333)      // Pardiso backward solve
#define PARDISO_FREE       ( -1)      // Free internal data structure

// Pardiso default parameters
static abip_int maxfct = 1;  // Maximum number of factors
static abip_int mnum   = 1;  // The matrix used for the solution phase
static abip_int mtype  = -2; // Real and symmetric indefinite
static abip_int msglvl = 0;  // Print no information
static abip_int idummy = 0;  // Dummy variable for taking up space
static abip_float ddummy = 0.0;  // Dummy variable for taking up space
```

```

// Pardiso solver
#define SYMBOLIC 3
#define PIVOTING 2
#define FACTORIZE 0

static abip_int PARDISO_PARAMS_LDL[PARDISOINDEX] = {

    1, /* Non-default value */ SYMBOLIC, /* P Nested dissection */ 0, /* Reserved          */
    0, /* No CG             */ 0, /* No user permutation */ 1, /* Overwriting    */
    0, /* Refinement report */ 0, /* Auto ItRef step     */ 0, /* Reserved          */
    8, /* Perturb           */ 0, /* Disable scaling     */ 0, /* No transpose      */
    0, /* Disable matching  */ 0, /* Report on pivots    */ 0, /* Output            */
    0, /* Output            */ 0, /* Output              */-1, /* No report         */
    0, /* No report         */ 0, /* Output              */ PIVOTING, /* Pivoting          */
    0, /* nPosEigVals       */ 0, /* nNegEigVals         */ FACTORIZE, /* Classic factorize */
    0,                         0,                           0, /* Matrix checker    */
    0,                         0,                           0,
    0,                         0,                           0,
    0,                         1, /* 0-based solve       */ 0,
    0,                         0,                           0,
    0,                         0,                           0,
    0,                         0,                           0,
    0,                         0,                           0,
    0,                         0,                           0,
    0,                         0,                           0,
    0,                         0, /* No diagonal         */ 0,
    0,                         0,                           0,
    0,                         0,                           0,
    0
};

#ifdef __cplusplus
extern "C" {
#endif

extern void pardisoinit ( void *, const abip_int *, abip_int * );

extern void pardiso     ( void     *, abip_int    *, abip_int *, abip_int *, abip_int *, abip_int *,
                          double   *, abip_int    *, abip_int *, abip_int *, abip_int *, abip_int *,
                          abip_int *, double      *, double   *, abip_int * );

extern abip_int pardisoFactorize( ABIPLinSysWork *p, cs *A );
extern void pardisoFree         ( ABIPLinSysWork *p, ABIPMatrix *A );
extern void pardisoSolve        ( ABIPLinSysWork *p, ABIPMatrix *A, abip_float *b );
#ifdef __cplusplus
}
#endif


#endif /* abip_pardiso_h */
```

5.72 linsys/amatrix.h File Reference

```
#include "glbopts.h"
```

Data Structures

- struct ABIP_A_DATA_MATRIX

5.73 amatrix.h

Go to the documentation of this file.

```
#ifndef AMATRIX_H_GUARD
#define AMATRIX_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

```

```
#include "glbopts.h"

struct ABIP_A_DATA_MATRIX
{
        abip_float *x;
        abip_int *i;
        abip_int *p;
        abip_int m;
        abip_int n;
};

#ifdef __cplusplus
}
#endif
#endif
```

5.74 linsys/common.c File Reference

```
#include "common.h"
#include "linsys.h"
```

Macros

- #define MIN_SCALE (1e-3)
- #define MAX_SCALE (1e3)

Functions

- abip_int ABIP() copy_A_matrix (ABIPMatrix **dstp, const ABIPMatrix *src)

    copy matrix A
- abip_int ABIP() validate_lin_sys (const ABIPMatrix *A)

    validate the linear system
- void ABIP() free_A_matrix (ABIPMatrix *A)

    set the memory of matrix A free
- void ABIP() _normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, ABIPScaling *scal)

    normalize matrix A
- void ABIP() _un_normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, const ABIPScaling *scal)

    unnormalize matrix A
- void ABIP() _accum_by_Atrans (abip_int n, abip_float *Ax, abip_int *Ai, abip_int *Ap, const abip_float *x, abip_float *y)

    compute A^Tx
- void ABIP() _accum_by_A (abip_int n, abip_float *Ax, abip_int *Ai, abip_int *Ap, const abip_float *x, abip_float *y)

    compute Ax
- abip_float ABIP() cumsum (abip_int *p, abip_int *c, abip_int n)

    compute cumulative sum of c

5.74.1 Macro Definition Documentation

5.74.1.1 MAX_SCALE

```
#define MAX_SCALE (1e3)
```

Definition at line 5 of file common.c.

5.74.1.2 MIN_SCALE

```
#define MIN_SCALE (1e-3)
```

Definition at line 4 of file common.c.

5.74.2 Function Documentation

5.74.2.1 _accum_by_A()

```
void ABIP() _accum_by_A (
            abip_int n,
            abip_float * Ax,
            abip_int * Ai,
            abip_int * Ap,
            const abip_float * x,
            abip_float * y )
```

compute Ax

Definition at line 644 of file common.c.

5.74.2.2 _accum_by_Atrans()

```
void ABIP() _accum_by_Atrans (
            abip_int n,
            abip_float * Ax,
            abip_int * Ai,
            abip_int * Ap,
            const abip_float * x,
            abip_float * y )
```

compute A^Tx

Definition at line 598 of file common.c.

5.74.2.3 _normalize_A()

```
void ABIP() _normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPScaling * scal )
```

normalize matrix A

Definition at line 150 of file common.c.

5.74.2.4 _un_normalize_A()

```
void ABIP() _un_normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            const ABIPScaling * scal )
```

unnormalize matrix A

Definition at line 569 of file common.c.

5.74.2.5 copy_A_matrix()

```
abip_int ABIP() copy_A_matrix (
            ABIPMatrix ** dstp,
            const ABIPMatrix * src )
```

copy matrix A

Definition at line 10 of file common.c.

5.74.2.6 cumsum()

```
abip_float ABIP() cumsum (
            abip_int * p,
            abip_int * c,
            abip_int n )
```

compute cumulative sum of c

Definition at line 700 of file common.c.

5.74.2.7 free_A_matrix()

```
void ABIP() free_A_matrix (
            ABIPMatrix * A )
```

set the memory of matrix A free

Definition at line 99 of file common.c.

5.74.2.8 validate_lin_sys()

```
abip_int ABIP() validate_lin_sys (
            const ABIPMatrix * A )
```

validate the linear system

Definition at line 45 of file common.c.

5.75 common.c

Go to the documentation of this file.

```
00001 #include "common.h"
00002 #include "linsys.h"
00003
00004 #define MIN_SCALE (1e-3)
00005 #define MAX_SCALE (1e3)
00006
00010 abip_int ABIP(copy_A_matrix)
00011 (
00012  ABIPMatrix **dstp,
00013  const ABIPMatrix *src
00014  )
00015 {
00016     abip_int Annz = src->p[src->n];
00017     ABIPMatrix *A = (ABIPMatrix *)abip_calloc(1, sizeof(ABIPMatrix));
00018     if (!A)
00019     {
00020         return 0;
00021     }
00022     A->n = src->n;
00023     A->m = src->m;
00024     A->x = (abip_float *) abip_malloc(sizeof(abip_float) * Annz);
00025     A->i = (abip_int *) abip_malloc(sizeof(abip_int) * Annz);
00026     A->p = (abip_int *) abip_malloc(sizeof(abip_int) * (src->n + 1));
00027
00028     if (!A->x || !A->i || !A->p)
00029     {
00030         abip_free(A->x); abip_free(A->i); abip_free(A->p); abip_free(A);
00031         return 0;
00032     }
00033
00034     memcpy(A->x, src->x, sizeof(abip_float) * Annz);
00035     memcpy(A->i, src->i, sizeof(abip_int) * Annz);
00036     memcpy(A->p, src->p, sizeof(abip_int) * (src->n + 1));
00037
00038     *dstp = A;
00039     return 1;
00040 }
00041
00045 abip_int ABIP(validate_lin_sys)
00046 (
00047  const ABIPMatrix *A
00048  )
00049 {
00050     abip_int i;
00051     abip_int r_max;
00052     abip_int Annz;
```

```
    if (!A->x || !A->i || !A->p)
    {
        abip_printf("ERROR: incomplete data!\n");
        return -1;
    }

    for (i = 0; i < A->n; ++i)
    {
        if (A->p[i] == A->p[i + 1])
        {
            abip_printf("WARN: the %li-th column empty!\n", (long)i);
        }
        else if (A->p[i] > A->p[i + 1])
        {
            abip_printf("ERROR: the column pointers decreases!\n");
            return -1;
        }
    }

    Annz = A->p[A->n];
    if (((abip_float)Annz / A->m > A->n) || (Annz <= 0))
    {
        abip_printf("ERROR: the number of nonzeros in A = %li, outside of valid range!\n", (long)
 Annz);
        return -1;
    }

    r_max = 0;
    for (i = 0; i < Annz; ++i)
    {
        if (A->i[i] > r_max)
        {
            r_max = A->i[i];
        }
    }
    if (r_max > A->m - 1)
    {
        abip_printf("ERROR: the number of rows in A is inconsistent with input dimension!\n");
        return -1;
    }

    return 0;
}
void ABIP(free_A_matrix)
(
 ABIPMatrix *A
 )
{
    if (A->x)
    {
        abip_free(A->x);
    }
    if (A->i)
    {
        abip_free(A->i);
    }
    if (A->p)
    {
        abip_free(A->p);
    }

    abip_free(A);
}

#if EXTRA_VERBOSE > 0

static void print_A_matrix
 (
  const ABIPMatrix *A
  )
{
    abip_int i;
    abip_int j;

    if (A->p[A->n] < 2500)
    {
        abip_printf("\n");
        for (i = 0; i < A->n; ++i)
        {
            abip_printf("Col %li: ", (long)i);
            for (j = A->p[i]; j < A->p[i + 1]; j++)
            {
                abip_printf("A[%li,%li] = %4f, ", (long)A->i[j], (long)i, A->x[j]);
            }
            abip_printf("norm col = %4f\n", ABIP(norm)(&(A->x[A->p[i]]), A->p[i + 1] - A->p[i]));
        }
```

```
        abip_printf("norm A = %4f\n", ABIP(norm)(A->x, A->p[A->n]));
    }
}
#endif

void ABIP(_normalize_A)
(
ABIPMatrix *A,
const ABIPSettings *stgs,
ABIPScaling *scal
)
{
    abip_float *D = (abip_float *) abip_malloc(A->m * sizeof(abip_float));
    abip_float *E = (abip_float *) abip_malloc(A->n * sizeof(abip_float));
    abip_float *nms = (abip_float *) abip_calloc(A->m, sizeof(abip_float));

    abip_float *D_pc = (abip_float *) abip_malloc(A->m * sizeof(abip_float)); // for pc rescale
    abip_float *E_pc = (abip_float *) abip_malloc(A->n * sizeof(abip_float));
    abip_float *D_origin = (abip_float *) abip_malloc(A->m * sizeof(abip_float));
    abip_float *E_origin = (abip_float *) abip_malloc(A->n * sizeof(abip_float));
    abip_float *D_temp = (abip_float *) abip_malloc(A->m * sizeof(abip_float));
    abip_float *E_temp = (abip_float *) abip_malloc(A->n * sizeof(abip_float));
    abip_float *D_ruiz = (abip_float *) abip_malloc(A->m * sizeof(abip_float)); // for ruiz rescale
    abip_float *E_ruiz = (abip_float *) abip_malloc(A->n * sizeof(abip_float));
    abip_float *D_qp = (abip_float *) abip_malloc(A->m * sizeof(abip_float)); // for the trial rescale
 used in qcp
    abip_float *E_qp = (abip_float *) abip_malloc(A->n * sizeof(abip_float));

    abip_float min_row_scale = MIN_SCALE * SQRTF((abip_float)A->n);
    abip_float max_row_scale = MAX_SCALE * SQRTF((abip_float)A->n);
    abip_float min_col_scale = MIN_SCALE * SQRTF((abip_float)A->m);
    abip_float max_col_scale = MAX_SCALE * SQRTF((abip_float)A->m);

    abip_int i;
    abip_int j;
    abip_int c1;
    abip_int c2;

    //
    abip_int k;
    abip_int ruiz_iter = stgs->ruiz_iter;
    abip_float tmp;
    // ------

    abip_float wrk;
    abip_float e;

#if EXTRA_VERBOSE > 0
    ABIP(timer) normalize_timer;
    ABIP(tic)(&normalize_timer);
    abip_printf("normalizing A\n");
    print_A_matrix(A);
#endif

    memset(D, 0, A->m * sizeof(abip_float));
    memset(E, 0, A->n * sizeof(abip_float));

    //
    memset(D_pc, 0, A->m * sizeof(abip_float));
    memset(E_pc, 0, A->n * sizeof(abip_float));
    memset(D_origin, 0, A->m * sizeof(abip_float));
    memset(E_origin, 0, A->n * sizeof(abip_float));
    memset(D_qp, 0, A->m * sizeof(abip_float));
    memset(E_qp, 0, A->n * sizeof(abip_float));

    for (i = 0; i < A->m; ++i){
        D_ruiz[i] = 1.0;
    }

    for (i = 0; i < A->n; ++i){
        E_ruiz[i] = 1.0;
    }

    if (stgs->pc_ruiz_rescale)
    {
        // for pc rescaling
        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i + 1] - A->p[i];
            e = ABIP(norm_one_sqrt)(&(A->x[A->p[i]]), c1); // compute sqrt of 1 norm
            if (e < min_col_scale){
                e = 1;
            }
            else if (e > max_col_scale){
                e = max_col_scale;
            }
            ABIP(scale_array)(&(A->x[A->p[i]]), 1.0 / e, c1);  // scale A
```

```
            E_pc[i] = e;
        }

        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i];
            c2 = A->p[i + 1];
            for (j = c1; j < c2; ++j)
            {
                wrk = A->x[j];
                D_pc[A->i[j]] += ABS(wrk);
            }
        }

        for (i = 0; i < A->m; ++i)
        {
            D_pc[i] = SQRTF(D_pc[i]);
            if (D_pc[i] < min_row_scale)
            {
                D_pc[i] = 1;
            }
            else if (D_pc[i] > max_row_scale)
            {
                D_pc[i] = max_row_scale;
            }
        }

        for (i = 0; i < A->n; ++i)
        {
            for (j = A->p[i]; j < A->p[i + 1]; ++j)
            {
                A->x[j] /= D_pc[A->i[j]];
            }
        }
        abip_printf("Done the pc rescaling!\n");
    }
    else
    {
        for (i = 0; i < A->m; ++i){
            D_pc[i] = 1.0;
        }

        for (i = 0; i < A->n; ++i){
            E_pc[i] = 1.0;
        }
    }

    // for origin rescaling
    if (stgs->origin_rescale)
    {
        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i + 1] - A->p[i];
            e = ABIP(norm)(&(A->x[A->p[i]]), c1);
            if (e < min_col_scale){
                e = 1;
            }
            else if (e > max_col_scale){
                e = max_col_scale;
            }
            ABIP(scale_array)(&(A->x[A->p[i]]), 1.0 / e, c1);  // scale A
            E_origin[i] = e;
        }

        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i];
            c2 = A->p[i + 1];
            for (j = c1; j < c2; ++j)
            {
                wrk = A->x[j];                              //  j-th nnz
                D_origin[A->i[j]] += wrk * wrk;             //  A->[i[j]] is the j-th nnz's row
            }
        }

        for (i = 0; i < A->m; ++i)
        {
            D_origin[i] = SQRTF(D_origin[i]);
            if (D_origin[i] < min_row_scale)
            {
                D_origin[i] = 1;
            }
            else if (D_origin[i] > max_row_scale)
            {
                D_origin[i] = max_row_scale;
            }
        }
```

```
        for (i = 0; i < A->n; ++i)
        {
            for (j = A->p[i]; j < A->p[i + 1]; ++j)
            {
                A->x[j] /= D_origin[A->i[j]];
            }
        }
        abip_printf("Done the origin rescaling!\n");
    }
    else
    {
        for (i = 0; i < A->m; ++i){
            D_origin[i] = 1.0;
        }

        for (i = 0; i < A->n; ++i){
            E_origin[i] = 1.0;
        }
    }

    if (stgs->pc_ruiz_rescale)
    {
        for(k = 0; k < ruiz_iter; ++k ){

            memset(D_temp, 0, A->m * sizeof(abip_float));
            memset(E_temp, 0, A->n * sizeof(abip_float));

            // find E_temp
            for (i = 0; i < A->n; ++i)
            {
                c1 = A->p[i + 1] - A->p[i];
                e = ABIP(norm_inf_sqrt)(&(A->x[A->p[i]]), c1);

                if (e < min_col_scale){
                    e = 1;
                }
                else if (e > max_col_scale){
                    e = max_col_scale;
                }

                ABIP(scale_array)(&(A->x[A->p[i]]), 1.0 / e, c1);  // scale A
                E_temp[i] = e;
            }

            // find D_temp
            for (i = 0; i < A->n; ++i)
            {
                c1 = A->p[i];
                c2 = A->p[i + 1];

                for (j = c1; j < c2; ++j)
                {
                    wrk = ABS(A->x[j]);   //abs j-th nnz

                    if (wrk >= D_temp[A->i[j]])
                    {
                        D_temp[A->i[j]] = wrk;
                    }
                }
            }

            for (i = 0; i < A->m; ++i)
            {
                D_temp[i] = SQRTF(D_temp[i]);

                if (D_temp[i] < min_row_scale)
                {
                    D_temp[i] = 1;
                }
                else if (D_temp[i] > max_row_scale)
                {
                    D_temp[i] = max_row_scale;
                }
            }

            for (i = 0; i < A->n; ++i)
            {
                for (j = A->p[i]; j < A->p[i + 1]; ++j)
                {
                    A->x[j] /= D_temp[A->i[j]];
                }
            }

            // record E_ruiz
            for (i = 0; i < A->n; ++i){
                E_ruiz[i] = E_ruiz[i] * E_temp[i];
```

```
            }
            // record D_ruiz
            for (i = 0; i < A->m; ++i){
                D_ruiz[i] = D_ruiz[i] * D_temp[i];
            }

        }
        abip_printf("Done the ruiz rescaling!\n");
    }

    if (stgs->qp_rescale)
    {
        memset(D_temp, 0, A->m * sizeof(abip_float));

        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i + 1] - A->p[i];
            e = ABIP(norm_inf)(&(A->x[A->p[i]]), c1); // compute the max abs
            tmp = ABIP(min_abs_sqrt)(&(A->x[A->p[i]]), c1, e); // compute the sqrt non-zero min
            e = tmp * SQRTF(e);

            // control E_qp
            if (e < min_col_scale){
                e = 1;
            }
            else if (e > max_col_scale){
                e = max_col_scale;
            }

            // abip_printf("1.0/e is: %f\n", 1.0 /e);

            ABIP(scale_array)(&(A->x[A->p[i]]), 1.0 / e, c1);  // scale A
            E_qp[i] = e;
        }

        // rescaling trick from QP. Find D_qp
        // find inf norm and min nonzero abs
        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i];
            c2 = A->p[i + 1];
            for (j = c1; j < c2; ++j)
            {
                wrk = ABS(A->x[j]);
                if (wrk >= D_qp[A->i[j]])
                {
                    D_qp[A->i[j]] = wrk;
                }
            }
        }

        // Initialize D_temp
        for (i = 0; i < A->m; ++i)
        {
            D_temp[i] = D_qp[i];
        }

        // compute the min nonzero abs and save it in D_temp
        for (i = 0; i < A->n; ++i)
        {
            c1 = A->p[i];
            c2 = A->p[i+1];
            for (j = c1; j < c2; ++j)
            {
                wrk = ABS(A->x[j]);
                if (wrk <= D_temp[A->i[j]] && wrk > 0){
                    D_temp[A->i[j]] = wrk;
                }
            }
        }

        // find and control D_qp

        for (i = 0; i < A->m; ++i)
        {
            D_qp[i] = SQRTF(D_qp[i] * D_temp[i]);
            if (D_qp[i] < min_row_scale)
            {
                D_qp[i] = 1;
            }
            else if (D_qp[i] > max_row_scale)
            {
                D_qp[i] = max_row_scale;
            }
        }

        for (i = 0; i < A->n; ++i)
```

```
        {
            for (j = A->p[i]; j < A->p[i + 1]; ++j)
            {
                A->x[j] /= D_qp[A->i[j]];
            }
        }
        abip_printf("Done the QP rescaling\n");
    }
    else
    {
        for (i = 0; i < A->m; ++i){
            D_qp[i] = 1.0;
        }

        for (i = 0; i < A->n; ++i){
            E_qp[i] = 1.0;
        }
    }

    // combine the above rescaling tricks
    for (i = 0; i < A->m; ++i)
    {
        D[i] = D_pc[i] * D_ruiz[i] * D_origin[i] *D_qp[i];
    }

    for (i = 0; i < A->n; ++i)
    {
        E[i] = E_pc[i] * E_ruiz[i] * E_origin[i] * E_qp[i];
    }
    // ---------------------------------------------------------

    for (i = 0; i < A->n; ++i)
    {
        for (j = A->p[i]; j < A->p[i + 1]; ++j)
        {
            wrk = A->x[j];
            nms[A->i[j]] += wrk * wrk;
        }
    }

    scal->mean_norm_row_A = 0.0;
    for (i = 0; i < A->m; ++i)
    {
        scal->mean_norm_row_A += SQRTF(nms[i]) / A->m;
    }

    abip_free(nms);

    scal->mean_norm_col_A = 0.0;
    for (i = 0; i < A->n; ++i)
    {
        c1 = A->p[i + 1] - A->p[i];
        scal->mean_norm_col_A+= ABIP(norm)(&(A->x[A->p[i]]), c1) / A->n;
    }

    if (stgs->scale != 1)
    {
        ABIP(scale_array)(A->x, stgs->scale, A->p[A->n]);
    }

    scal->D = D;
    scal->E = E;

    abip_free(nms);
    abip_free(D_pc); abip_free(E_pc); abip_free(D_origin);
    abip_free(E_origin); abip_free(D_temp); abip_free(E_temp);
    abip_free(D_ruiz); abip_free(E_ruiz); abip_free(D_qp); abip_free(E_qp);

#if EXTRA_VERBOSE > 0
    abip_printf("finished normalizing A, time: %1.2e seconds. \n", ABIP(tocq)(&normalize_timer) /
   1e3);
    print_A_matrix(A);
#endif

}
void ABIP(_un_normalize_A)
(
 ABIPMatrix *A,
 const ABIPSettings *stgs,
 const ABIPScaling *scal
 )
{
    abip_int i;
    abip_int j;

    abip_float *D = scal->D;
    abip_float *E = scal->E;
```

```
    for (i = 0; i < A->n; ++i)
    {
        for (j = A->p[i]; j < A->p[i + 1]; ++j)
        {
            A->x[j] *= D[A->i[j]];
        }
    }

    for (i = 0; i < A->n; ++i)
    {
        ABIP(scale_array)(&(A->x[A->p[i]]), E[i] / stgs->scale, A->p[i + 1] - A->p[i]);
    }
}
void ABIP(_accum_by_Atrans)
(
 abip_int n,
 abip_float *Ax,
 abip_int *Ai,
 abip_int *Ap,
 const abip_float *x,
 abip_float *y
 )
{
    abip_int p;
    abip_int j;

    abip_int c1;
    abip_int c2;
    abip_float yj;

#if EXTRA_VERBOSE > 0
    ABIP(timer) mult_by_Atrans_timer;
    ABIP(tic)(&mult_by_Atrans_timer);
#endif

#ifdef _OPENMP
#pragma omp parallel for private(p, c1, c2, yj)
#endif

    for (j = 0; j < n; j++)
    {
        yj = y[j];
        c1 = Ap[j];
        c2 = Ap[j + 1];
        for (p = c1; p < c2; p++)
        {
            yj += Ax[p] * x[Ai[p]];
        }
        y[j] = yj;
    }

#if EXTRA_VERBOSE > 0
    abip_printf("mult By A trans time: %1.2e seconds. \n", ABIP(tocq)(&mult_by_Atrans_timer) / 1e3);
#endif
}

void ABIP(_accum_by_A)
(
 abip_int n,
 abip_float *Ax,
 abip_int *Ai,
 abip_int *Ap,
 const abip_float *x,
 abip_float *y
 )
{
    abip_int p;
    abip_int j;

    abip_int c1;
    abip_int c2;
    abip_float xj;

#if EXTRA_VERBOSE > 0
    ABIP(timer) mult_by_A_timer;
    ABIP(tic)(&mult_by_A_timer);
#endif

#ifdef _OPENMP
#pragma omp parallel for private(p, c1, c2, xj)
    for (j = 0; j < n; j++)
    {
        xj = x[j];
        c1 = Ap[j];
        c2 = Ap[j + 1];
        for (p = c1; p < c2; p++)
```

```
        {
#pragma omp atomic
            y[Ai[p]] += Ax[p] * xj;
        }
    }
#endif

    for (j = 0; j < n; j++)
    {
        xj = x[j];
        c1 = Ap[j];
        c2 = Ap[j + 1];
        for (p = c1; p < c2; p++)
        {
            y[Ai[p]] += Ax[p] * xj;
        }
    }

#if EXTRA_VERBOSE > 0
    abip_printf("mult By A time: %1.2e seconds \n", ABIP(tocq)(&mult_by_A_timer) / 1e3);
#endif
}

abip_float ABIP(cumsum)
(
 abip_int *p,
 abip_int *c,
 abip_int n
 )
{
    abip_int i;
    abip_float nz = 0;
    abip_float nz2 = 0;

    if (!p || !c)
    {
        return (-1);
    }

    for (i = 0; i < n; i++)
    {
        p[i] = nz;
        nz += c[i];
        nz2 += c[i];
        c[i] = p[i];
    }

    p[n] = nz;
    return nz2;
}
```

5.76 linsys/common.h File Reference

```
#include "abip.h"
#include "amatrix.h"
#include "linsys.h"
#include "linalg.h"
#include "util.h"
```

Functions

- void ABIP() _accum_by_Atrans (abip_int n, abip_float *Ax, abip_int *Ai, abip_int *Ap, const abip_float *x, abip_float *y)

  compute A^Tx
- void ABIP() _accum_by_A (abip_int n, abip_float *Ax, abip_int *Ai, abip_int *Ap, const abip_float *x, abip_float *y)

  compute Ax
- void ABIP() _normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, ABIPScaling *scal)

  normalize matrix A

- void ABIP() _un_normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, const ABIPScaling *scal)
  unnormalize matrix A
- abip_float ABIP() cumsum (abip_int *p, abip_int *c, abip_int n)
  compute cumulative sum of c

5.76.1 Function Documentation

5.76.1.1 _accum_by_A()

```
void ABIP() _accum_by_A (
            abip_int n,
            abip_float * Ax,
            abip_int * Ai,
            abip_int * Ap,
            const abip_float * x,
            abip_float * y )
```

compute Ax

Definition at line 644 of file common.c.

5.76.1.2 _accum_by_Atrans()

```
void ABIP() _accum_by_Atrans (
            abip_int n,
            abip_float * Ax,
            abip_int * Ai,
            abip_int * Ap,
            const abip_float * x,
            abip_float * y )
```

compute A^Tx

Definition at line 598 of file common.c.

5.76.1.3 _normalize_A()

```
void ABIP() _normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPScaling * scal )
```

normalize matrix A

Definition at line 150 of file common.c.

5.76.1.4 _un_normalize_A()

```
void ABIP() _un_normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            const ABIPScaling * scal )
```

unnormalize matrix A

Definition at line 569 of file common.c.

5.76.1.5 cumsum()

```
abip_float ABIP() cumsum (
            abip_int * p,
            abip_int * c,
            abip_int n )
```

compute cumulative sum of c

Definition at line 700 of file common.c.

5.77 common.h

Go to the documentation of this file.

```
#ifndef COMMON_H_GUARD
#define COMMON_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "abip.h"
#include "amatrix.h"
#include "linsys.h"
#include "linalg.h"
#include "util.h"

void ABIP(_accum_by_Atrans)
(
    abip_int n,
    abip_float *Ax,
    abip_int *Ai,
    abip_int *Ap,
    const abip_float *x,
    abip_float *y
);

void ABIP(_accum_by_A)
(
    abip_int n,
    abip_float *Ax,
    abip_int *Ai,
    abip_int *Ap,
    const abip_float *x,
    abip_float *y
);

void ABIP(_normalize_A)
(
    ABIPMatrix *A,
    const ABIPSettings *stgs,
    ABIPScaling *scal
);

```

```
void ABIP(_un_normalize_A)
(
    ABIPMatrix *A,
    const ABIPSettings *stgs,
    const ABIPScaling *scal
);

abip_float ABIP(cumsum)
(
    abip_int *p,
    abip_int *c,
    abip_int n
);

#ifdef __cplusplus
}
#endif

#endif
```

5.78 linsys/direct.c File Reference

```
#include "direct.h"
```

Functions

- char *ABIP() get_lin_sys_method (const ABIPMatrix *A, const ABIPSettings *stgs)
- char *ABIP() get_lin_sys_summary (ABIPLinSysWork *p, const ABIPInfo *info)
- void ABIP() free_lin_sys_work (ABIPLinSysWork *p)
- cs * form_kkt (const ABIPMatrix *A, const ABIPSettings *s)
- abip_int _ldl_init (cs *A, abip_int P[ ], abip_float **info)
- abip_int _ldl_factor (cs *A, abip_int P[ ], abip_int Pinv[ ], cs **L, abip_float **D)
- void _ldl_solve (abip_float *x, abip_float b[ ], cs *L, abip_float D[ ], abip_int P[ ], abip_float *bp)
- void ABIP() accum_by_Atrans (const ABIPMatrix *A, ABIPLinSysWork *p, const abip_float *x, abip_float *y)
- void ABIP() accum_by_A (const ABIPMatrix *A, ABIPLinSysWork *p, const abip_float *x, abip_float *y)
- void ABIP() normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, ABIPScaling *scal)
- void ABIP() un_normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, const ABIPScaling *scal)
- abip_int factorize (const ABIPMatrix *A, const ABIPSettings *stgs, ABIPLinSysWork *p)
- ABIPLinSysWork *ABIP() init_lin_sys_work (const ABIPMatrix *A, const ABIPSettings *stgs)
- abip_int ABIP() solve_lin_sys (const ABIPMatrix *A, const ABIPSettings *stgs, ABIPLinSysWork *p, abip_float *b, const abip_float *s, abip_int iter)

5.78.1 Function Documentation

5.78.1.1 _ldl_factor()

```
abip_int _ldl_factor (
            cs * A,
            abip_int P[ ],
            abip_int Pinv[ ],
            cs ** L,
            abip_float ** D )
```

Definition at line 121 of file direct.c.

5.78.1.2 _ldl_init()

```
abip_int _ldl_init (
            cs * A,
            abip_int P[ ],
            abip_float ** info )
```

Definition at line 106 of file direct.c.

5.78.1.3 _ldl_solve()

```
void _ldl_solve (
            abip_float * x,
            abip_float b[ ],
            cs * L,
            abip_float D[ ],
            abip_int P[ ],
            abip_float * bp )
```

Definition at line 172 of file direct.c.

5.78.1.4 accum_by_A()

```
void ABIP() accum_by_A (
            const ABIPMatrix * A,
            ABIPLinSysWork * p,
            const abip_float * x,
            abip_float * y )
```

Definition at line 205 of file direct.c.

5.78.1.5 accum_by_Atrans()

```
void ABIP() accum_by_Atrans (
            const ABIPMatrix * A,
            ABIPLinSysWork * p,
            const abip_float * x,
            abip_float * y )
```

Definition at line 200 of file direct.c.

5.78.1.6 factorize()

```
abip_int factorize (
            const ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPLinSysWork * p )
```

Definition at line 218 of file direct.c.

5.78.1.7 form_kkt()

```
cs * form_kkt (
            const ABIPMatrix * A,
            const ABIPSettings * s )
```

Definition at line 49 of file direct.c.

5.78.1.8 free_lin_sys_work()

```
void ABIP() free_lin_sys_work (
            ABIPLinSysWork * p )
```

Definition at line 28 of file direct.c.

5.78.1.9 get_lin_sys_method()

```
char *ABIP() get_lin_sys_method (
            const ABIPMatrix * A,
            const ABIPSettings * stgs )
```

Definition at line 5 of file direct.c.

5.78.1.10 get_lin_sys_summary()

```
char *ABIP() get_lin_sys_summary (
            ABIPLinSysWork * p,
            const ABIPInfo * info )
```

Definition at line 15 of file direct.c.

5.78.1.11 init_lin_sys_work()

```
ABIPLinSysWork *ABIP() init_lin_sys_work (
            const ABIPMatrix * A,
            const ABIPSettings * stgs )
```

Definition at line 273 of file direct.c.

5.78.1.12 normalize_A()

```
void ABIP() normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPScaling * scal )
```

Definition at line 210 of file direct.c.

5.78.1.13 solve_lin_sys()

```
abip_int ABIP() solve_lin_sys (
            const ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPLinSysWork * p,
            abip_float * b,
            const abip_float * s,
            abip_int iter )
```

Definition at line 305 of file direct.c.

5.78.1.14 un_normalize_A()

```
void ABIP() un_normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            const ABIPScaling * scal )
```

Definition at line 214 of file direct.c.

5.79 direct.c

Go to the documentation of this file.

```c
#include "direct.h"

// use ldl to solve the linear system

char *ABIP(get_lin_sys_method) ( const ABIPMatrix *A, const ABIPSettings *stgs ) {
    char *tmp = (char *) abip_malloc(sizeof(char) * 128);
#ifdef ABIP_PARDISO
    sprintf(tmp, "sparse-direct-intel-pardiso, nnz in A = %li", (long)A->p[A->n]);
#else
    sprintf(tmp, "sparse-direct, nnz in A = %li", (long)A->p[A->n]);
#endif
    return tmp;
}

char *ABIP(get_lin_sys_summary) ( ABIPLinSysWork *p, const ABIPInfo *info ) {
    char *str = (char *) abip_malloc(sizeof(char) * 128);
#ifdef ABIP_PARDISO
    sprintf(str, "Linear system avg solve time: %1.2es\n", p->total_solve_time / (info->admm_iter + 1) / 1e3);
#else
    abip_int n = p->L->n;
    sprintf(str, "\tLin-sys: nnz in L factor: %li, avg solve time: %1.2es\n",
            (long)(p->L->p[n] + n), p->total_solve_time / (info->admm_iter + 1) / 1e3);
#endif

    p->total_solve_time = 0; return str;
}

void ABIP(free_lin_sys_work) ( ABIPLinSysWork *p ) {
    if (p) {
        if (p->L)  { ABIP(cs_spfree)(p->L); }
        if (p->P)  { abip_free(p->P); }
        if (p->bp) { abip_free(p->bp); }
        if (p->D)  { abip_free(p->D); }
        abip_free(p);
    }
}

#ifdef ABIP_PARDISO
void ABIP(free_lin_sys_work_pds) ( ABIPLinSysWork *p, ABIPMatrix *A ) {
    if (p) {
        pardisoFree(p, A);
        if (p->P)  { abip_free(p->D); }
        if (p->D)  { abip_free(p->D); }
        abip_free(p);
    }
}
#endif

cs *form_kkt ( const ABIPMatrix *A, const ABIPSettings *s ) {

    abip_int j, k, kk; cs *K_cs;
    const abip_int Annz = A->p[A->n];
    const abip_int Knnzmax = A->m + A->n + Annz;

    cs *K = ABIP(cs_spalloc)(A->m + A->n, A->m + A->n, Knnzmax, 1, 1);

#if EXTRA_VERBOSE > 0
    abip_printf("forming KKT\n");
#endif

    if (!K)
    {
        return ABIP_NULL;
    }

    kk = 0;
    for (k = 0; k < A->m; k++)
    {
        K->i[kk] = k;
        K->p[kk] = k;
        K->x[kk] = s->rho_y;
        kk++;
    }

    for (j = 0; j < A->n; j++)
    {
        for (k = A->p[j]; k < A->p[j + 1]; k++)
        {
            K->p[kk] = j + A->m;
            K->i[kk] = A->i[k];
            K->x[kk] = A->x[k];
```

```c
                kk++;
            }
        }
        for (k = 0; k < A->n; k++)
        {
            K->i[kk] = k + A->m;
            K->p[kk] = k + A->m;
            K->x[kk] = -1;
            kk++;
        }

        K->nnz = Knnzmax;
        K_cs = ABIP(cs_compress)(K);

#ifdef ABIP_PARDISO
        cs *KT = ABIP(cs_transpose)(K_cs, 1);
        ABIP(cs_spfree)(K); ABIP(cs_spfree)(K_cs);
        return (KT);
#else
        ABIP(cs_spfree)(K);
        return (K_cs);
#endif
}

abip_int _ldl_init ( cs *A, abip_int P[], abip_float **info ) {
#ifdef ABIP_PARDISO
        return 0;
#else
        *info = (abip_float *)abip_malloc(AMD_INFO * sizeof(abip_float));

#ifdef DLONG
        return (amd_l_order(A->n, A->p, A->i, P, (abip_float *) ABIP_NULL, *info));
#else
        return (amd_order(A->n, A->p, A->i, P, (abip_float *) ABIP_NULL, *info));
#endif

#endif
}

abip_int _ldl_factor ( cs *A, abip_int P[], abip_int Pinv[], cs **L, abip_float **D ) {

#ifdef ABIP_PARDISO
        return 0;
#else
        abip_int kk;
        abip_int n = A->n;

        abip_int *Parent = (abip_int *)abip_malloc(n * sizeof(abip_int));
        abip_int *Lnnz = (abip_int *)abip_malloc(n * sizeof(abip_int));
        abip_int *Flag = (abip_int *)abip_malloc(n * sizeof(abip_int));
        abip_int *Pattern = (abip_int *)abip_malloc(n * sizeof(abip_int));
        abip_float *Y = (abip_float *)abip_malloc(n * sizeof(abip_float));
        (*L)->p = (abip_int *)abip_malloc((1 + n) * sizeof(abip_int));

        /*abip_int Parent[n], Lnz[n], Flag[n], Pattern[n]; */
        /*abip_float Y[n]; */

        LDL_symbolic(n, A->p, A->i, (*L)->p, Parent, Lnnz, Flag, P, Pinv);

        (*L)->nnzmax = *((*L)->p + n);
        (*L)->x = (abip_float *)abip_malloc((*L)->nnzmax * sizeof(abip_float));
        (*L)->i = (abip_int *)abip_malloc((*L)->nnzmax * sizeof(abip_int));
        *D = (abip_float *)abip_malloc(n * sizeof(abip_float));

        if (!(*D) || !(*L)->i || !(*L)->x || !Y || !Pattern || !Flag || !Lnnz || !Parent)
        {
            abip_free(Pattern); abip_free(Y);
            return -1;
        }

#if EXTRA_VERBOSE > 0
        abip_printf("numeric factorization\n");
#endif

        kk = LDL_numeric(n, A->p, A->i, A->x, (*L)->p, Parent, Lnnz, (*L)->i, (*L)->x, *D, Y, Pattern, Flag, P, Pinv);

#if EXTRA_VERBOSE > 0
        abip_printf("finished numeric factorization\n");
#endif

        abip_free(Parent);
        abip_free(Lnnz);
        abip_free(Flag);
        abip_free(Pattern);
        abip_free(Y);
        return (kk - n);
```

```
#endif

}

void _ldl_solve ( abip_float *x, abip_float b[], cs *L, abip_float D[],
                  abip_int P[], abip_float *bp ) {
#ifdef ABIP_PARDISO
    return;
#else
    abip_int n = L->n;
    if (P == ABIP_NULL)
    {
        if (x != b)
        {
            memcpy(x, b, n * sizeof(abip_float));
        }

        LDL_lsolve(n, x, L->p, L->i, L->x);
        LDL_dsolve(n, x, D);
        LDL_ltsolve(n, x, L->p, L->i, L->x);
    }
    else
    {
        LDL_perm(n, bp, b, P);
        LDL_lsolve(n, bp, L->p, L->i, L->x);
        LDL_dsolve(n, bp, D);
        LDL_ltsolve(n, bp, L->p, L->i, L->x);
        LDL_permt(n, x, bp, P);
    }
#endif
}

void ABIP(accum_by_Atrans) ( const ABIPMatrix *A, ABIPLinSysWork *p,
                             const abip_float *x, abip_float *y ) {
    ABIP(_accum_by_Atrans)(A->n, A->x, A->i, A->p, x, y);
}

void ABIP(accum_by_A) ( const ABIPMatrix *A, ABIPLinSysWork *p,
                        const abip_float *x, abip_float *y ) {
    ABIP(_accum_by_A)(A->n, A->x, A->i, A->p, x, y);
}

void ABIP(normalize_A) ( ABIPMatrix *A, const ABIPSettings *stgs, ABIPScaling *scal ) {
    ABIP(_normalize_A)(A, stgs, scal);
}

void ABIP(un_normalize_A) ( ABIPMatrix *A, const ABIPSettings *stgs, const ABIPScaling *scal ) {
    ABIP(_un_normalize_A)(A, stgs, scal);
}

abip_int factorize ( const ABIPMatrix *A, const ABIPSettings *stgs, ABIPLinSysWork *p ) {

#ifdef ABIP_PARDISO
    abip_int ret_code = 0;
    cs *K = form_kkt(A, stgs);
    if (!K) { return -1; }
    ret_code = pardisoFactorize(p, K);
    ABIP(cs_spfree)(K);
    return ret_code;
#else
    abip_float *info;
    abip_int *Pinv;
    abip_int amd_status;
    abip_int ldl_status;

    cs *C;
    cs *K = form_kkt(A, stgs);
    if (!K)
    {
        return -1;
    }

    amd_status = _ldl_init(K, p->P, &info);
    if (amd_status < 0)
    {
        return (amd_status);
    }

#if EXTRA_VERBOSE > 0
    if (stgs->verbose)
    {
        abip_printf("Matrix factorization info:\n");

#ifdef DLONG
        amd_l_info(info);
#else
        amd_info(info);
```

```
#endif
    }
#endif
    Pinv = ABIP(cs_pinv)(p->P, A->m + A->n);
    C = ABIP(cs_symperm)(K, Pinv, 1);
    ldl_status = _ldl_factor(C, ABIP_NULL, ABIP_NULL, &p->L, &p->D);
    ABIP(cs_spfree)(C);
    ABIP(cs_spfree)(K);
    abip_free(Pinv);
    abip_free(info);
    return (ldl_status);
#endif
}
ABIPLinSysWork *ABIP(init_lin_sys_work)
(
 const ABIPMatrix *A,
 const ABIPSettings *stgs
 )
{
    ABIPLinSysWork *p = (ABIPLinSysWork *) abip_calloc(1, sizeof(ABIPLinSysWork));
    abip_int m_plus_n = A->m + A->n;
#ifdef ABIP_PARDISO
    p->P = (abip_int *) abip_malloc(sizeof(abip_int) * m_plus_n);
    p->D = (abip_float *) abip_malloc(sizeof(abip_float) * m_plus_n);
    if (factorize(A, stgs, p)) {
        ABIP(free_lin_sys_work)(p); return ABIP_NULL;
    }
#else
    p->P = (abip_int *) abip_malloc(sizeof(abip_int) * m_plus_n);
    p->L = (cs *) abip_malloc(sizeof(cs));
    p->bp = (abip_float *) abip_malloc(m_plus_n * sizeof(abip_float));
    p->L->m = m_plus_n;
    p->L->n = m_plus_n;
    p->L->nnz = -1;
    if (factorize(A, stgs, p) < 0)
    {
        ABIP(free_lin_sys_work)(p);
        return ABIP_NULL;
    }
#endif
    p->total_solve_time = 0.0;
    return p;
}
abip_int ABIP(solve_lin_sys)
(
 const ABIPMatrix *A,
 const ABIPSettings *stgs,
 ABIPLinSysWork *p,
 abip_float *b,
 const abip_float *s,
 abip_int iter
 )
{
    ABIP(timer) linsys_timer;
    ABIP(tic)(&linsys_timer);
#ifdef ABIP_PARDISO
    pardisoSolve(p, (ABIPMatrix *) A, b);
#else
    _ldl_solve(b, b, p->L, p->D, p->P, p->bp);
#endif
    p->total_solve_time += ABIP(tocq)(&linsys_timer);
#if EXTRA_VERBOSE > 0
    abip_printf("linsys solve time: %1.2es\n", ABIP(tocq)(&linsys_timer) / 1e3);
#endif
    return 0;
}
```

5.80 linsys/direct.h File Reference

```
#include "glbopts.h"
#include "abip.h"
#include "cs.h"
```

```
#include "amd.h"
#include "ldl.h"
#include "common.h"
#include "abip_pardiso.h"
```

Data Structures

- struct ABIP_LIN_SYS_WORK

5.81 direct.h

Go to the documentation of this file.

```
#ifndef PRIV_H_GUARD
#define PRIV_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include "glbopts.h"
#include "abip.h"
#include "cs.h"
#include "amd.h"
#include "ldl.h"
#include "common.h"
#include "abip_pardiso.h"

struct ABIP_LIN_SYS_WORK
{
    cs *L;
    abip_float *D;
    abip_int *P;
    abip_int *i;
    abip_int *j;
    abip_float *bp;

    void *pardiso_work[PARDISOINDEX];
    abip_float total_solve_time;
};

#ifdef __cplusplus
}
#endif
#endif
```

5.82 linsys/indirect.c File Reference

```
#include "indirect.h"
```

Macros

- #define CG_BEST_TOL 1e-9
- #define CG_MIN_TOL 1e-1

Functions

- char *ABIP() get_lin_sys_method (const ABIPMatrix *A, const ABIPSettings *stgs)
- char *ABIP() get_lin_sys_summary (ABIPLinSysWork *p, const ABIPInfo *info)
- void ABIP() free_lin_sys_work (ABIPLinSysWork *p)
- void ABIP() accum_by_Atrans (const ABIPMatrix *A, ABIPLinSysWork *p, const abip_float *x, abip_float *y)
- void ABIP() accum_by_A (const ABIPMatrix *A, ABIPLinSysWork *p, const abip_float *x, abip_float *y)
- void ABIP() normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, ABIPScaling *scal)
- void ABIP() un_normalize_A (ABIPMatrix *A, const ABIPSettings *stgs, const ABIPScaling *scal)
- ABIPLinSysWork *ABIP() init_lin_sys_work (const ABIPMatrix *A, const ABIPSettings *stgs)
- abip_int ABIP() solve_lin_sys (const ABIPMatrix *A, const ABIPSettings *stgs, ABIPLinSysWork *p, abip_float *b, const abip_float *s, abip_int iter)

5.82.1 Macro Definition Documentation

5.82.1.1 CG_BEST_TOL

```
#define CG_BEST_TOL 1e-9
```

Definition at line 3 of file indirect.c.

5.82.1.2 CG_MIN_TOL

```
#define CG_MIN_TOL 1e-1
```

Definition at line 4 of file indirect.c.

5.82.2 Function Documentation

5.82.2.1 accum_by_A()

```
void ABIP() accum_by_A (
            const ABIPMatrix * A,
            ABIPLinSysWork * p,
            const abip_float * x,
            abip_float * y )
```

Definition at line 233 of file indirect.c.

5.82.2.2 accum_by_Atrans()

```
void ABIP() accum_by_Atrans (
            const ABIPMatrix * A,
            ABIPLinSysWork * p,
            const abip_float * x,
            abip_float * y )
```

Definition at line 222 of file indirect.c.

5.82.2.3 free_lin_sys_work()

```
void ABIP() free_lin_sys_work (
            ABIPLinSysWork * p )
```

Definition at line 141 of file indirect.c.

5.82.2.4 get_lin_sys_method()

```
char *ABIP() get_lin_sys_method (
            const ABIPMatrix * A,
            const ABIPSettings * stgs )
```

Definition at line 8 of file indirect.c.

5.82.2.5 get_lin_sys_summary()

```
char *ABIP() get_lin_sys_summary (
            ABIPLinSysWork * p,
            const ABIPInfo * info )
```

Definition at line 20 of file indirect.c.

5.82.2.6 init_lin_sys_work()

```
ABIPLinSysWork *ABIP() init_lin_sys_work (
            const ABIPMatrix * A,
            const ABIPSettings * stgs )
```

Definition at line 282 of file indirect.c.

5.82.2.7 normalize_A()

```
void ABIP() normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPScaling * scal )
```

Definition at line 244 of file indirect.c.

5.82.2.8 solve_lin_sys()

```
abip_int ABIP() solve_lin_sys (
            const ABIPMatrix * A,
            const ABIPSettings * stgs,
            ABIPLinSysWork * p,
            abip_float * b,
            const abip_float * s,
            abip_int iter )
```

Definition at line 393 of file indirect.c.

5.82.2.9 un_normalize_A()

```
void ABIP() un_normalize_A (
            ABIPMatrix * A,
            const ABIPSettings * stgs,
            const ABIPScaling * scal )
```

Definition at line 253 of file indirect.c.

5.83 indirect.c

Go to the documentation of this file.

```
#include "indirect.h"

#define CG_BEST_TOL 1e-9
#define CG_MIN_TOL 1e-1

// use cg to solve the linear system

char *ABIP(get_lin_sys_method)
(
 const ABIPMatrix *A,
 const ABIPSettings *stgs
 )
{
    char *str = (char *)abip_malloc(sizeof(char) * 128);
    sprintf(str, "sparse-indirect, nnz in A = %li, CG tol ~ 1/iter^(%2.2f)",
            (long)A->p[A->n], stgs->cg_rate);
    return str;
}

char *ABIP(get_lin_sys_summary)
(
 ABIPLinSysWork *p,
```

```
 const ABIPInfo *info
 )
{
    char *str = (char *)abip_malloc(sizeof(char) * 128);
    sprintf(str, "\tLin-sys: avg # CG iterations: %2.2f, avg solve time: %1.2es\n",
            (abip_float)p->tot_cg_its / (info->admm_iter + 1),
            p->total_solve_time / (info->admm_iter + 1) / 1e3);
    p->tot_cg_its = 0;
    p->total_solve_time = 0;
    return str;
}

/* M = inv(diag(RHO_Y * I + AA')) */
static void get_preconditioner
 (
  const ABIPMatrix *A,
  const ABIPSettings *stgs,
  ABIPLinSysWork *p
  )
{
    abip_int i;
    abip_int j;
    abip_int c1;
    abip_int c2;

    abip_float wrk;
    abip_float *M = p->M;

    memset(M, 0, A->m * sizeof(abip_float));

#if EXTRA_VERBOSE > 0
    abip_printf("getting pre-conditioner\n");
#endif

    for (i = 0; i < A->n; ++i)
    {
        c1 = A->p[i];
        c2 = A->p[i + 1];
        for (j = c1; j < c2; ++j)
        {
            wrk = A->x[j];
            M[A->i[j]] += wrk * wrk;
        }
    }

    for (i = 0; i < A->m; ++i)
    {
        M[i] = 1 / M[i];
        /* M[i] = 1; */
    }

#if EXTRA_VERBOSE > 0
    ABIP(print_array)(M, A->m, "M");
    abip_printf("norm M = %4f\n", ABIP(norm)(M, A->m));
    abip_printf("finished getting pre-conditioner\n");
#endif
}

static void transpose
 (
  const ABIPMatrix *A,
  ABIPLinSysWork *p
  )
{
    abip_int *Ci = p->At->i;
    abip_int *Cp = p->At->p;
    abip_float *Cx = p->At->x;

    abip_int m = A->m;
    abip_int n = A->n;

    abip_int *Ap = A->p;
    abip_int *Ai = A->i;
    abip_float *Ax = A->x;

    abip_int i;
    abip_int j;
    abip_int q;
    abip_int *z;
    abip_int c1;
    abip_int c2;

#if EXTRA_VERBOSE > 0
    ABIP(timer) transpose_timer;
    abip_printf("transposing A\n");
    ABIP(tic)(&transpose_timer);
#endif
```

```
    z = (abip_int *)abip_calloc(m, sizeof(abip_int));

    for (i = 0; i < Ap[n]; i++)
    {
        z[Ai[i]]++;             /* row counts */
    }

    ABIP(cumsum)(Cp, z, m);      /* row pointers */

    for (j = 0; j < n; j++)
    {
        c1 = Ap[j];
        c2 = Ap[j + 1];
        for (i = c1; i < c2; i++)
        {
            q = z[Ai[i]];
            Ci[q] = j;          /* place A(i,j) as entry C(j,i) */
            Cx[q] = Ax[i];
            z[Ai[i]]++;
        }
    }

    abip_free(z);

#if EXTRA_VERBOSE > 0
    abip_printf("finished transposing A, time: %1.2es\n",
                ABIP(tocq)(&transpose_timer) / 1e3);
#endif
}

void ABIP(free_lin_sys_work)
(
 ABIPLinSysWork *p
 )
{
    if (p)
    {
        if (p->p)
        {
            abip_free(p->p);
        }

        if (p->r)
        {
            abip_free(p->r);
        }

        if (p->Gp)
        {
            abip_free(p->Gp);
        }

        if (p->tmp)
        {
            abip_free(p->tmp);
        }

        if (p->At)
        {
            ABIPMatrix *At = p->At;

            if (At->i)
            {
                abip_free(At->i);
            }

            if (At->x)
            {
                abip_free(At->x);
            }

            if (At->p)
            {
                abip_free(At->p);
            }

            abip_free(At);
        }

        if (p->z)
        {
            abip_free(p->z);
        }

        if (p->M)
        {
```

```
            abip_free(p->M);
        }

        abip_free(p);
    }
}

/*y = (RHO_Y * I + AA')x */
static void mat_vec
 (
 const ABIPMatrix *A,
 const ABIPSettings *s,
 ABIPLinSysWork *p,
 const abip_float *x,
 abip_float *y
 )
{
    abip_float *tmp = p->tmp;
    memset(tmp, 0, A->n * sizeof(abip_float));
    ABIP(accum_by_Atrans)(A, p, x, tmp);
    memset(y, 0, A->m * sizeof(abip_float));
    ABIP(accum_by_A)(A, p, tmp, y);
    ABIP(add_scaled_array)(y, x, A->m, s->rho_y);
}

void ABIP(accum_by_Atrans)
(
const ABIPMatrix *A,
ABIPLinSysWork *p,
const abip_float *x,
abip_float *y
)
{
    ABIP(_accum_by_Atrans)(A->n, A->x, A->i, A->p, x, y);
}

void ABIP(accum_by_A)
(
const ABIPMatrix *A,
ABIPLinSysWork *p,
const abip_float *x,
abip_float *y
)
{
    ABIP(_accum_by_Atrans)(p->At->n, p->At->x, p->At->i, p->At->p, x, y);
}

void ABIP(normalize_A)
(
ABIPMatrix *A,
const ABIPSettings *stgs,
ABIPScaling *scal)
{
    ABIP(_normalize_A)(A, stgs, scal);
}

void ABIP(un_normalize_A)
(
ABIPMatrix *A,
const ABIPSettings *stgs,
const ABIPScaling *scal
)
{
    ABIP(_un_normalize_A)(A, stgs, scal);
}

static void apply_pre_conditioner
 (
 abip_float *M,
 abip_float *z,
 abip_float *r,
 abip_int m,
 abip_float *ipzr
 )
{
    abip_int i;
    *ipzr = 0;

    for (i = 0; i < m; ++i)
    {
        z[i] = r[i] * M[i];
        *ipzr += z[i] * r[i];
    }
}

ABIPLinSysWork *ABIP(init_lin_sys_work)
(
```

```
 const ABIPMatrix *A,
 const ABIPSettings *stgs
 )
{
    ABIPLinSysWork *p = (ABIPLinSysWork *)abip_calloc(1, sizeof(ABIPLinSysWork));
    p->p = (abip_float *)abip_malloc((A->m) * sizeof(abip_float));
    p->r = (abip_float *)abip_malloc((A->m) * sizeof(abip_float));
    p->Gp = (abip_float *)abip_malloc((A->m) * sizeof(abip_float));
    p->tmp = (abip_float *)abip_malloc((A->n) * sizeof(abip_float));

    /* memory for A transpose */
    p->At = (ABIPMatrix *)abip_malloc(sizeof(ABIPMatrix));
    p->At->m = A->n;
    p->At->n = A->m;
    p->At->i = (abip_int *)abip_malloc((A->p[A->n]) * sizeof(abip_int));
    p->At->p = (abip_int *)abip_malloc((A->m + 1) * sizeof(abip_int));
    p->At->x = (abip_float *)abip_malloc((A->p[A->n]) * sizeof(abip_float));
    transpose(A, p);

    /* preconditioner memory */
    p->z = (abip_float *)abip_malloc((A->m) * sizeof(abip_float));
    p->M = (abip_float *)abip_malloc((A->m) * sizeof(abip_float));
    get_preconditioner(A, stgs, p);

    p->total_solve_time = 0;
    p->tot_cg_its = 0;

    if (!p->p || !p->r || !p->Gp || !p->tmp || !p->At || !p->At->i || !p->At->p || !p->At->x)
    {
        ABIP(free_lin_sys_work)(p);
        return ABIP_NULL;
    }

    return p;
}

/* solves (I+AA')x = b, s warm start, solution stored in b */
static abip_int pcg
 (
  const ABIPMatrix *A,
  const ABIPSettings *stgs,
  ABIPLinSysWork *pr,
  const abip_float *s,
  abip_float *b,
  abip_int max_its,
  abip_float tol
  )
{
    abip_int i;
    abip_int m = A->m;

    abip_float ipzr;
    abip_float ipzr_old;
    abip_float alpha;

    abip_float *p = pr->p;   /* cg direction */
    abip_float *Gp = pr->Gp; /* updated CG direction */
    abip_float *r = pr->r;   /* cg residual */
    abip_float *z = pr->z;   /* for preconditioning */
    abip_float *M = pr->M;   /* inverse diagonal preconditioner */

    if (s == ABIP_NULL)
    {
        memcpy(r, b, m * sizeof(abip_float));
        memset(b, 0, m * sizeof(abip_float));
    }
    else
    {
        mat_vec(A, stgs, pr, s, r);
        ABIP(add_scaled_array)(r, b, m, -1);
        ABIP(scale_array)(r, -1, m);
        memcpy(b, s, m * sizeof(abip_float));
    }

    /* check to see if we need to run CG at all */
    if (ABIP(norm)(r, m) < MIN(tol, 1e-18))
    {
        return 0;
    }

    apply_pre_conditioner(M, z, r, m, &ipzr);
    memcpy(p, z, m * sizeof(abip_float));

    /* main loop */
    for (i = 0; i < max_its; ++i)
    {
        mat_vec(A, stgs, pr, p, Gp);
```

```
        alpha = ipzr / ABIP(dot)(p, Gp, m);
        ABIP(add_scaled_array)(b, p, m, alpha);
        ABIP(add_scaled_array)(r, Gp, m, -alpha);

        if (ABIP(norm)(r, m) < tol)
        {
#if EXTRA_VERBOSE > 0
            abip_printf("tol: %.4e, resid: %.4e, iters: %li\n", tol, ABIP(norm)(r, m), (long)i + 1);
#endif

            return i + 1;
        }

        ipzr_old = ipzr;
        apply_pre_conditioner(M, z, r, m, &ipzr);
        ABIP(scale_array)(p, ipzr / ipzr_old, m);
        ABIP(add_scaled_array)(p, z, m, 1);
    }

    return i;
}

abip_int ABIP(solve_lin_sys)
(
 const ABIPMatrix *A,
 const ABIPSettings *stgs,
 ABIPLinSysWork *p,
 abip_float *b,
 const abip_float *s,
 abip_int iter
 )
{
    abip_int cg_its;
    ABIP(timer) linsys_timer;

    abip_float cg_tol = ABIP(norm)(b, A->m) * (iter < 0 ? CG_BEST_TOL
                                                    : CG_MIN_TOL / POWF((abip_float)iter + 1,
 stgs->cg_rate));

    cg_tol = MAX(cg_tol, 1e-07);

    ABIP(tic)(&linsys_timer);

    /* solves Mx = b, for x but stores result in b */
    /* s contains warm-start (if available) */
    ABIP(accum_by_A)(A, p, &(b[A->m]), b);

    /* solves (I+AA')x = b, s warm start, solution stored in b */
    cg_its = pcg(A, stgs, p, s, b, A->m, MAX(cg_tol, CG_BEST_TOL));
    ABIP(scale_array)(&(b[A->m]), -1, A->n);
    ABIP(accum_by_Atrans)(A, p, b, &(b[A->m]));

    if (iter >= 0)
    {
        p->tot_cg_its += cg_its;
    }

    p->total_solve_time += ABIP(tocq)(&linsys_timer);

#if EXTRA_VERBOSE > 0
    abip_printf("linsys solve time: %1.2es\n", ABIP(tocq)(&linsys_timer) / 1e3);
#endif

    return 0;
}
```

5.84 linsys/indirect.h File Reference

```
#include <math.h>
#include "common.h"
#include "glbopts.h"
#include "linalg.h"
#include "abip.h"
```

Data Structures

- struct ABIP_LIN_SYS_WORK

5.85 indirect.h

Go to the documentation of this file.

```
#ifndef PRIV_H_GUARD
#define PRIV_H_GUARD

#ifdef __cplusplus
extern "C" {
#endif

#include <math.h>
#include "common.h"
#include "glbopts.h"
#include "linalg.h"
#include "abip.h"

struct ABIP_LIN_SYS_WORK
{
    abip_float *p;     /* cg iterate  */
    abip_float *r;     /* cg residual */
    abip_float *Gp;
    abip_float *tmp;
    ABIPMatrix *At;

    /* preconditioning */
    abip_float *z;
    abip_float *M;

    /* reporting */
    abip_int tot_cg_its;
    abip_float total_solve_time;
};

#ifdef __cplusplus
}
#endif
#endif
```

5.86 make_abip.m File Reference

Typedefs

- using int = false

Functions

- if (~isempty(strfind(computer, '64'))) flags.arr
- end if (isunix &&~ismac) flags.link
- elseif (ismac) flags.link
- end if (float) flags.LCFLAG
- end if (int) flags.INT
- end if (flags.COMPILE_WITH_OPENMP) flags.link
- end delete (fullfile(".", "interface", "*."+mexext))
- compile_direct (flags, common_abip)
- compile_indirect (flags, common_abip)
- movefile (fullfile(".", "*."+mexext), fullfile(".", "interface"))
- addpath (fullfile("mexfile"))
- addpath (fullfile("interface"))
- addpath (fullfile("test"))

Variables

- parameter setting for compiling the ABIP solver gpu = false
- compile the gpu version of ABIP float = false
- use bit integers for indexing WARNING
- use bit integers for indexing use with caution openmp parallelizes the matrix multiply for the indirect solver(using CG) flags EXTRA_VERBOSE = 0
- flags BLASLIB = ''
- MATLAB_MEX_FILE env variable sets blasint to ptrdiff_t flags LCFLAG = '-DMATLAB_MEX_FILE -DUSE↩ _LAPACK -DCTRLC=1 -DCOPYAMATRIX'
- flags INCS = ''
- flags LOCS = ''
- Common source files abip_common_src = ["linalg.c"
- adaptive c
- abip_common_linsys = ["common.c"]
- abip_mexfile = ["abip_mex.c"]
- common_abip = strcat(abip_common_src, " ", abip_common_linsys, " ", abip_mexfile)
- else flags arr = ''
- else flags link = '-lut'
- else flags INT = '-DDLONG'

5.86.1 Typedef Documentation

5.86.1.1 int

```
using int = false
```

Definition at line 4 of file make_abip.m.

5.86.2 Function Documentation

5.86.2.1 addpath() [1/3]

```
addpath (
            fullfile("interface")   )
```

5.86.2.2 addpath() [2/3]

```
addpath (
            fullfile("mexfile")   )
```

5.86.2.3 addpath() [3/3]

```
addpath (
            fullfile("test")  )
```

5.86.2.4 compile_direct()

```
compile_direct (
            flags ,
            common_abip  )
```

5.86.2.5 compile_indirect()

```
compile_indirect (
            flags ,
            common_abip  )
```

5.86.2.6 delete()

```
end delete (
            fullfile(".", "interface", "*."+mexext)  )
```

5.86.2.7 elseif()

```
elseif (
            ismac  )
```

5.86.2.8 if() [1/5]

```
end if (
            flags.  COMPILE_WITH_OPENMP )
```

5.86.2.9 if() [2/5]

```
end if (
            float  )
```

5.86.2.10 if() [3/5]

```
end if (
            int  )
```

5.86.2.11 if() [4/5]

```
end if (
            isunix &&~ ismac )
```

5.86.2.12 if() [5/5]

```
if (
            ~ isemptystrfind(computer, '64') )
```

5.86.2.13 movefile()

```
movefile (
            fullfile(".", "*."+mexext) ,
            fullfile(".", "interface")  )
```

5.86.3 Variable Documentation

5.86.3.1 abip_common_linsys

```
abip_common_linsys = ["common.c"]
```

Definition at line 21 of file make_abip.m.

5.86.3.2 abip_common_src

```
abip_common_src = ["linalg.c"
```

Definition at line 19 of file make_abip.m.

5.86.3.3 abip_mexfile

```
abip_mexfile = ["abip_mex.c"]
```

Definition at line 22 of file make_abip.m.

5.86.3.4 arr

```
else flags arr = ''
```

Definition at line 34 of file make_abip.m.

5.86.3.5 BLASLIB

```
flags BLASLIB = ''
```

Definition at line 11 of file make_abip.m.

5.86.3.6 c

```
abip_version c
```

Definition at line 19 of file make_abip.m.

5.86.3.7 common_abip

```
common_abip = strcat(abip_common_src, " ", abip_common_linsys, " ", abip_mexfile)
```

Definition at line 29 of file make_abip.m.

5.86.3.8 EXTRA_VERBOSE

```
use bit integers for indexing use with caution openmp parallelizes the matrix multiply for the
indirect solver (using CG) flags EXTRA_VERBOSE = 0
```

Definition at line 9 of file make_abip.m.

5.86.3.9 float

```
compile the gpu version of ABIP float = false
```

Definition at line 3 of file make_abip.m.

5.86.3.10 gpu

```
parameter setting for compiling the ABIP solver gpu = false
```

Definition at line 2 of file make_abip.m.

5.86.3.11 INCS

```
flags INCS = ''
```

Definition at line 14 of file make_abip.m.

5.86.3.12 INT

```
else flags INT = '-DDLONG'
```

Definition at line 53 of file make_abip.m.

5.86.3.13 LCFLAG

```
flags LCFLAG = '-DMATLAB_MEX_FILE -DUSE_LAPACK -DCTRLC=1 -DCOPYAMATRIX'
```

Definition at line 13 of file make_abip.m.

5.86.3.14 link

```
else flags link = '-lut'
```

Definition at line 42 of file make_abip.m.

5.86.3.15 LOCS

```
flags LOCS = ''
```

Definition at line 15 of file make_abip.m.

5.86.3.16 WARNING

```
use bit integers for indexing WARNING
```

Definition at line 6 of file make_abip.m.

5.87 make_abip.m

Go to the documentation of this file.

```
00001 %% parameter setting for compiling the ABIP solver.
00002 gpu = false;         % compile the gpu version of ABIP.
00003 float = false;          % using single precision (rather than double) floating points
00004 int = false;            % use 32 bit integers for indexing
00005
00006 % WARNING: openmp used in MATLAB can cause errors and crashes, use with caution.
00007 % openmp parallelizes the matrix multiply for the indirect solver (using CG):
00008 flags.COMPILE_WITH_OPENMP = false;
00009 flags.EXTRA_VERBOSE = 0;
00010
00011 flags.BLASLIB = ”;
00012 % MATLAB_MEX_FILE env variable sets blasint to ptrdiff_t.
00013 flags.LCFLAG  = ’-DMATLAB_MEX_FILE -DUSE_LAPACK -DCTRLC=1 -DCOPYAMATRIX’;
00014 flags.INCS = ”;
00015 flags.LOCS = ”;
00016
00017
00018 % Common source files
00019 abip_common_src = ["linalg.c"; "adaptive.c"; "cs.c"; "util.c"; "abip.c";
00020                     "ctrlc.c"; "normalize.c"; "abip_version.c"];
00021 abip_common_linsys = ["common.c"];
00022 abip_mexfile = ["abip_mex.c"];
00023
00024 abip_common_src = fullfile(’src’, abip_common_src);
00025 abip_common_src = strjoin(abip_common_src);
00026 abip_common_linsys = fullfile(’linsys’, abip_common_linsys);
00027 abip_mexfile = fullfile(’mexfile’, abip_mexfile);
00028
00029 common_abip = strcat(abip_common_src, " ", abip_common_linsys, " ", abip_mexfile);
00030
00031 if (~isempty (strfind (computer, ’64’)))
00032     flags.arr = ’-largeArrayDims’;
00033 else
00034     flags.arr = ”;
00035 end
00036
00037 if (isunix && ~ismac)
00038     flags.link = ’-lm -lut -lrt’;
00039 elseif  (ismac)
00040     flags.link = ’-lm -lut’;
00041 else
00042     flags.link = ’-lut’;
00043     flags.LCFLAG = sprintf(’-DNOBLASSUFFIX %s’, flags.LCFLAG);
00044 end
00045
00046 if (float)
00047     flags.LCFLAG = sprintf(’-DSFLOAT %s’, flags.LCFLAG);
00048 end
00049
00050 if (int)
00051     flags.INT = ”;
00052 else
00053     flags.INT = ’-DDLONG’;
00054 end
00055
```

```
if (flags.COMPILE_WITH_OPENMP)
    flags.link = strcat(flags.link, ' -lgomp');
end

if (flags.EXTRA_VERBOSE)
    flags.LCFLAG = sprintf('-DEXTRA_VERBOSE %s', flags.LCFLAG);
end

delete(fullfile(".", "interface", "*." + mexext));
compile_direct(flags, common_abip);
compile_indirect(flags, common_abip);
movefile(fullfile(".", "*." + mexext), fullfile(".", "interface"));
addpath(fullfile("mexfile"));
addpath(fullfile("interface"));
% addpath(fullfile("test"));
savepath
```

5.88 mexfile/abip_mex.c File Reference

```
#include "glbopts.h"
#include "linalg.h"
#include "amatrix.h"
#include "matrix.h"
#include "mex.h"
#include "abip.h"
#include "util.h"
```

Functions

- void free_mex (ABIPData ∗d)

  set memory free
- abip_int parse_warm_start (const mxArray ∗p_mex, abip_float ∗∗p, abip_int len)
- abip_int ∗ cast_to_abip_int_arr (mwIndex ∗arr, abip_int len)
- void set_output_field (mxArray ∗∗pout, abip_float ∗out, abip_int len)
- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])

  obtain data from mex function

5.88.1 Function Documentation

5.88.1.1 cast_to_abip_int_arr()

```
abip_int * cast_to_abip_int_arr (
            mwIndex * arr,
            abip_int len )
```

Definition at line 31 of file abip_mex.c.

5.88.1.2 free_mex()

```
void free_mex (
            ABIPData * d )
```

set memory free

Definition at line 429 of file abip_mex.c.

5.88.1.3 mexFunction()

```
void mexFunction (
            int nlhs,
            mxArray * plhs[ ],
            int nrhs,
            const mxArray * prhs[ ] )
```

obtain data from mex function

Definition at line 83 of file abip_mex.c.

5.88.1.4 parse_warm_start()

```
abip_int parse_warm_start (
            const mxArray * p_mex,
            abip_float ** p,
            abip_int len )
```

Definition at line 11 of file abip_mex.c.

5.88.1.5 set_output_field()

```
void set_output_field (
            mxArray ** pout,
            abip_float * out,
            abip_int len )
```

Definition at line 67 of file abip_mex.c.

5.89 abip_mex.c

Go to the documentation of this file.

```
#include "glbopts.h"
#include "linalg.h"
#include "amatrix.h"
#include "matrix.h"
#include "mex.h"
#include "abip.h"
#include "util.h"

void free_mex(ABIPData *d);

abip_int parse_warm_start(const mxArray *p_mex, abip_float **p, abip_int len)
{
      *p = (abip_float *)abip_calloc(len, sizeof(abip_float));
      if (p_mex == ABIP_NULL)
      {
            return 0;
      }
      else if (mxIsSparse(p_mex) || (abip_int) *mxGetDimensions(p_mex) != len)
      {
            abip_printf("Error in warm-start (the input vectors should be dense and of correct size), running without full warm-start");
            return 0;
      }
      else
      {
            memcpy(*p, mxGetPr(p_mex), len * sizeof(abip_float));
            return 1;
      }
}

#if !(DLONG > 0)
abip_int *cast_to_abip_int_arr(mwIndex *arr, abip_int len)
{
      abip_int i;
      abip_int *arr_out = (abip_int *)abip_malloc(sizeof(abip_int) * len);
      for (i = 0; i < len; i++)
      {
            arr_out[i] = (abip_int) arr[i];
      }
      return arr_out;
}
#endif

#if SFLOAT > 0
abip_float *cast_to_abip_float_arr(double *arr, abip_int len)
{
      abip_int i;
      abip_float *arr_out = (abip_float *)abip_malloc(sizeof(abip_float) * len);
      for (i = 0; i < len; i++)
      {
            arr_out[i] = (abip_float) arr[i];
      }
      return arr_out;
}

double *cast_to_double_arr(abip_float *arr, abip_int len)
{
      abip_int i;
      double *arr_out = (double *)abip_malloc(sizeof(double) * len);
      for (i = 0; i < len; i++)
      {
            arr_out[i] = (double) arr[i];
      }
      return arr_out;
}
#endif

void set_output_field(mxArray **pout, abip_float *out, abip_int len)
{
      *pout = mxCreateDoubleMatrix(0, 0, mxREAL);
      #if SFLOAT > 0
            mxSetPr(*pout, cast_to_double_arr(out, len));
            abip_free(out);
      #else
            mxSetPr(*pout, out);
      #endif
      mxSetM(*pout, len);
      mxSetN(*pout, 1);
}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
```

```
    abip_int i;
    abip_int status;

    ABIPData *d;
    ABIPSolution sol = {0};
    ABIPInfo info;
    ABIPMatrix *A;

    const mxArray *data;
    const mxArray *A_mex;
    const mxArray *b_mex;
    const mxArray *c_mex;

    const mxArray *settings;

    const mwSize one[1] = {1};
    const int num_info_fields = 13;
    const char *info_fields[] = {"status", "ipm_iter", "admm_iter", "mu", "pobj", "dobj", "resPri",
"resDual", "relGap","resInfeas", "resUnbdd", "setupTime", "solveTime"};

    mxArray *tmp;

    #if EXTRA_VERBOSE > 0
        abip_printf("size of mwSize = %i\n", (int) sizeof(mwSize));
        abip_printf("size of mwIndex = %i\n", (int) sizeof(mwIndex));
    #endif

    if (nrhs != 2)
    {
        mexErrMsgTxt("Inputs are required in this order: data struct, settings struct");
    }

    if (nlhs > 4)
    {
        mexErrMsgTxt("ABIP returns up to 4 output arguments only.");
    }

    d = (ABIPData *)mxMalloc(sizeof(ABIPData));
    d->stgs = (ABIPSettings *)mxMalloc(sizeof(ABIPSettings));
    data = prhs[0];

    A_mex = (mxArray *)mxGetField(data, 0, "A");

    if (A_mex == ABIP_NULL)
    {
        abip_free(d);
        mexErrMsgTxt("ABIPData struct must contain a matrix 'A'.");
    }

    if (!mxIsSparse(A_mex))
    {
        abip_free(d);
        mexErrMsgTxt("Input matrix A must be in sparse format.");
    }

    b_mex = (mxArray *)mxGetField(data, 0, "b");

    if (b_mex == ABIP_NULL)
    {
        abip_free(d);
        mexErrMsgTxt("ABIPData struct must contain a vector 'b'.");
    }

    if (mxIsSparse(b_mex))
    {
        abip_free(d);
        mexErrMsgTxt("Input vector b must be in dense format.");
    }

    c_mex = (mxArray *)mxGetField(data, 0, "c");

    if (c_mex == ABIP_NULL)
    {
        abip_free(d);
        mexErrMsgTxt("ABIPData struct must contain a vector 'c'.");
    }

    if (mxIsSparse(c_mex))
    {
        abip_free(d);
        mexErrMsgTxt("Input vector c must be in dense format.");
    }

    settings = prhs[1];

    d->n = (abip_int) * (mxGetDimensions(c_mex));
    d->m = (abip_int) * (mxGetDimensions(b_mex));
```

```
#if SFLOAT > 0
        d->b = castTo_abip_float_arr(mxGetPr(b_mex), d->m);
        d->c = cast_to_abip_float_arr(mxGetPr(c_mex), d->n);
#else
        d->b = (abip_float *)mxGetPr(b_mex);
        d->c = (abip_float *)mxGetPr(c_mex);
#endif

// set default parameters
ABIP(set_default_settings)(d);

tmp = mxGetField(settings, 0, "max_ipm_iters");
if (tmp != ABIP_NULL)
{
        d->stgs->max_ipm_iters = (abip_int)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "max_admm_iters");
if (tmp != ABIP_NULL)
{
        d->stgs->max_admm_iters = (abip_int)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "eps");
if (tmp != ABIP_NULL)
{
        d->stgs->eps = (abip_float)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "cg_rate");
if (tmp != ABIP_NULL)
{
        d->stgs->cg_rate = (abip_float)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "alpha");
if (tmp != ABIP_NULL)
{
        d->stgs->alpha = (abip_float)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "rho_y");
if (tmp != ABIP_NULL)
{
        d->stgs->rho_y = (abip_float)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "normalize");
if (tmp != ABIP_NULL)
{
        d->stgs->normalize = (abip_int)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "scale");
if (tmp != ABIP_NULL)
{
        d->stgs->scale = (abip_float)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "sparsity_ratio");
if (tmp != ABIP_NULL)
{
        d->stgs->sparsity_ratio = (abip_float)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "adaptive");
if (tmp != ABIP_NULL)
{
        d->stgs->adaptive = (abip_int)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "adaptive_lookback");
if (tmp != ABIP_NULL)
{
        d->stgs->adaptive_lookback = (abip_int)*mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "dynamic_sigma");
if (tmp != ABIP_NULL) {
        d->stgs->dynamic_sigma = (abip_float) *mxGetPr(tmp);
}

tmp = mxGetField(settings, 0, "dynamic_x");
if (tmp != ABIP_NULL) {
        d->stgs->dynamic_x = (abip_float) *mxGetPr(tmp);
}
```

```
    tmp = mxGetField(settings, 0, "dynamic_eta");
    if (tmp != ABIP_NULL) {
        d->stgs->dynamic_eta = (abip_float) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "restart_thresh");
    if (tmp != ABIP_NULL) {
        d->stgs->restart_thresh = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "restart_fre");
    if (tmp != ABIP_NULL) {
        d->stgs->restart_fre = (abip_float) *mxGetPr(tmp);
    }

    // add by Kurt. 22.05.03
    tmp = mxGetField(settings, 0, "origin_rescale");
    if (tmp != ABIP_NULL) {
        d->stgs->origin_rescale = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "pc_ruiz_rescale");
    if (tmp != ABIP_NULL) {
        d->stgs->pc_ruiz_rescale = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "qp_rescale");
    if (tmp != ABIP_NULL) {
        d->stgs->qp_rescale = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "ruiz_iter");
    if (tmp != ABIP_NULL) {
        d->stgs->ruiz_iter = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "hybrid_mu");
    if (tmp != ABIP_NULL) {
        d->stgs->hybrid_mu = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "half_update");
    if (tmp != ABIP_NULL) {
        d->stgs->half_update = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "avg_criterion");
    if (tmp != ABIP_NULL) {
        d->stgs->avg_criterion = (abip_int) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "hybrid_thresh");
    if (tmp != ABIP_NULL) {
        d->stgs->hybrid_thresh = (abip_float) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "dynamic_sigma_second");
    if (tmp != ABIP_NULL) {
        d->stgs->dynamic_sigma_second = (abip_float) *mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "timelimit");
    if (tmp != ABIP_NULL) {
      d->stgs->max_time = (abip_float)*mxGetPr(tmp);
    }
    else {
      d->stgs->max_time = 3600;
    }
    // ------------

    tmp = mxGetField(settings, 0, "verbose");
    if (tmp != ABIP_NULL)
    {
        d->stgs->verbose = (abip_int)*mxGetPr(tmp);
    }

    tmp = mxGetField(settings, 0, "feasopt");
    if (tmp != ABIP_NULL)
    {
        d->stgs->pfeasopt = (abip_int)*mxGetPr(tmp);
    } else {
        d->stgs->pfeasopt = 0;
    }

    A = (ABIPMatrix *)abip_malloc(sizeof(ABIPMatrix));
    A->n = d->n;
```

```
    A->m = d->m;

    #if DLONG > 0
        A->p = (abip_int *)mxGetJc(A_mex);
        A->i = (abip_int *)mxGetIr(A_mex);
    #else
        A->p = cast_to_abip_int_arr(mxGetJc(A_mex), A->n + 1);
        A->i = cast_to_abip_int_arr(mxGetIr(A_mex), A->p[A->n]);
    #endif

    #if SFLOAT > 0
        A->x = cast_to_abip_float_arr(mxGetPr(A_mex), A->p[A->n]);
    #else
        A->x = (abip_float *)mxGetPr(A_mex);
    #endif

    d->A = A;
    d->sp = (abip_float)A->p[A->n]/(A->m*A->n);

    d->stgs->warm_start = parse_warm_start((mxArray *) mxGetField(data, 0, "x"), &(sol.x), d->n);
    d->stgs->warm_start |= parse_warm_start((mxArray *) mxGetField(data, 0, "y"), &(sol.y), d->m);
    d->stgs->warm_start |= parse_warm_start((mxArray *) mxGetField(data, 0, "s"), &(sol.s), d->n);

    status = ABIP(main)(d, &sol, &info);

    set_output_field(&plhs[0], sol.x, d->n);
    set_output_field(&plhs[1], sol.y, d->m);
    set_output_field(&plhs[2], sol.s, d->n);

    plhs[3] = mxCreateStructArray(1, one, num_info_fields, info_fields);

    mxSetField(plhs[3], 0, "status", mxCreateString(info.status));

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "ipm_iter", tmp);
    *mxGetPr(tmp) = (abip_float)info.ipm_iter;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "admm_iter", tmp);
    *mxGetPr(tmp) = (abip_float)info.admm_iter;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "pobj", tmp);
    *mxGetPr(tmp) = info.pobj;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "dobj", tmp);
    *mxGetPr(tmp) = info.dobj;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "resPri", tmp);
    *mxGetPr(tmp) = info.res_pri;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "resDual", tmp);
    *mxGetPr(tmp) = info.res_dual;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "relGap", tmp);
    *mxGetPr(tmp) = info.rel_gap;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "resInfeas", tmp);
    *mxGetPr(tmp) = info.res_infeas;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "resUnbdd", tmp);
    *mxGetPr(tmp) = info.res_unbdd;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "setupTime", tmp);
    *mxGetPr(tmp) = info.setup_time;

    tmp = mxCreateDoubleMatrix(1, 1, mxREAL);
    mxSetField(plhs[3], 0, "solveTime", tmp);
    *mxGetPr(tmp) = info.solve_time;

    free_mex(d);
    return;
}

void free_mex(ABIPData *d)
{
    if (d)
    {
        #if SFLOAT > 0
        if (d->b)
```

```
            {
                abip_free(d->b);
             }
            if (d->c)
            {
                abip_free(d->c);
            }
            #endif

            if (d->A)
            {
                #if !(DLONG > 0)
                if (d->A->p)
                {
                    abip_free(d->A->p);
                }
                if (d->A->i)
                {
                    abip_free(d->A->i);
                }
                #endif

                #if SFLOAT > 0
                if (d->A->x)
                {
                    abip_free(d->A->x);
                }
                #endif

                abip_free(d->A);
            }

            if (d->stgs)
            {
                abip_free(d->stgs);
            }

            abip_free(d);
        }
}
```

5.90 mexfile/abip_version_mex.c File Reference

```
#include "mex.h"
#include "abip.h"
```

Functions

- void mexFunction (int nlhs, mxArray ∗plhs[ ], int nrhs, const mxArray ∗prhs[ ])

5.90.1 Function Documentation

5.90.1.1 mexFunction()

```
void mexFunction (
        int nlhs,
        mxArray * plhs[ ],
        int nrhs,
        const mxArray * prhs[ ] )
```

Definition at line 4 of file abip_version_mex.c.

5.91 abip_version_mex.c

Go to the documentation of this file.

```
#include "mex.h"
#include "abip.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
    if (nrhs != 0) {
        mexErrMsgTxt("Too many input arguments.");
    }
    if (nlhs > 1) {
        mexErrMsgTxt("Too many output arguments.");
    }
    plhs[0] = mxCreateString(abip_version());
    return;
}
```

5.92 src/abip.c File Reference

```
#include <assert.h>
#include <time.h>
#include "abip.h"
#include "glbopts.h"
#include "adaptive.h"
#include "ctrlc.h"
#include "linalg.h"
#include "linsys.h"
#include "normalize.h"
#include "util.h"
```

Functions

- ABIP (timer)
- abip_int ABIP() solve (ABIPWork *w, const ABIPData *d, ABIPSolution *sol, ABIPInfo *info)

  detailed update rule of ABIP
- void ABIP() finish (ABIPWork *w)
- ABIPWork *ABIP() init (const ABIPData *d, ABIPInfo *info)
- abip_int ABIP() main (const ABIPData *d, ABIPSolution *sol, ABIPInfo *info)

  the main function

5.92.1 Function Documentation

5.92.1.1 ABIP()

```
ABIP (
            timer )
```

Definition at line 12 of file abip.c.

5.92.1.2 finish()

```
void ABIP() finish (
            ABIPWork * w )
```

Definition at line 2301 of file abip.c.

5.92.1.3 init()

```
ABIPWork *ABIP() init (
            const ABIPData * d,
            ABIPInfo * info )
```

Definition at line 2341 of file abip.c.

5.92.1.4 main()

```
abip_int ABIP() main (
            const ABIPData * d,
            ABIPSolution * sol,
            ABIPInfo * info )
```

the main function

Definition at line 2393 of file abip.c.

5.92.1.5 solve()

```
abip_int ABIP() solve (
            ABIPWork * w,
            const ABIPData * d,
            ABIPSolution * sol,
            ABIPInfo * info )
```

detailed update rule of ABIP

Definition at line 2056 of file abip.c.

5.93 abip.c

Go to the documentation of this file.

```c
#include <assert.h>
#include <time.h>
#include "abip.h"
#include "glbopts.h"
#include "adaptive.h"
#include "ctrlc.h"
#include "linalg.h"
#include "linsys.h"
#include "normalize.h"
#include "util.h"

ABIP(timer) global_timer;

/*
@brief printing header
*/
static const char *HEADER[] = {
    " ipm iter ", " admm iter ", "      mu ",
    " pri res ", " dua res ", " rel gap ",
    " pri obj ", " dua obj ", " kap/tau ", " time (s)",
};

static const abip_int HSPACE = 9;
static const abip_int HEADER_LEN = 10;
static const abip_int LINE_LEN = 150;

static abip_int abip_isnan
  (
   abip_float x
  )
{
    DEBUG_FUNC
    RETURN(x == NAN || x != x);
}
static void free_work
  (
   ABIPWork *w
  )
{
    DEBUG_FUNC

    if (!w)  {
        RETURN;
    }

    if (w->u)
    {
        abip_free(w->u);
    }

    if (w->v)
    {
        abip_free(w->v);
    }

    if (w->u_t)
    {
        abip_free(w->u_t);
    }

    if (w->u_avg)
    {
        abip_free(w->u_avg);
    }

    if (w->v_avg)
    {
        abip_free(w->v_avg);
    }

    if (w->u_avgcon)
    {
        abip_free(w->u_avgcon);
    }

    if (w->v_avgcon)
    {
        abip_free(w->v_avgcon);
    }

    if (w->u_sumcon)
    {
```

```
        abip_free(w->u_sumcon);
    }

    if (w->v_sumcon)
    {
        abip_free(w->v_sumcon);
    }

    if (w->u_prev)
    {
        abip_free(w->u_prev);
    }

    if (w->v_prev)
    {
        abip_free(w->v_prev);
    }

    if (w->h)
    {
        abip_free(w->h);
    }

    if (w->g)
    {
        abip_free(w->g);
    }

    if (w->pr)
    {
        abip_free(w->pr);
    }

    if (w->dr)
    {
        abip_free(w->dr);
    }

    if (w->b)
    {
        abip_free(w->b);
    }

    if (w->c)
    {
        abip_free(w->c);
    }

    if (w->scal)
    {
        if (w->scal->D)
        {
            abip_free(w->scal->D);
        }

        if (w->scal->E)
        {
            abip_free(w->scal->E);
        }

        abip_free(w->scal);
    }

    abip_free(w);

    RETURN;
}
static void print_init_header
 (
  const ABIPData *d
  )
{
    DEBUG_FUNC

    abip_int i;
    ABIPSettings *stgs = d->stgs;
    char *lin_sys_method = ABIP(get_lin_sys_method)(d->A, d->stgs);

    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("-");
    }

    abip_printf("\n\tABIP v%s - First-Order Interior-Point Solver\n\t(c) Tianyi Lin, UC Berkeley,
 2018-2020\n\t(c)LEAVES Group, 2021-2024\n",
                 ABIP(version)());

```

```
    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("-");
    }

    abip_printf("\n");

    if (lin_sys_method)
    {
        abip_printf("Lin-sys: %s\n", lin_sys_method);
        abip_free(lin_sys_method);
    }

    if (stgs->normalize)
    {
        abip_printf("eps = %.2e, alpha = %.2f, max_ipm_iters = %i, max_admm_iters = %i, normalize = %i\n"
                        "scale = %2.2f, adaptive = %i, adaptive_lookback = %i, rho_y = %.2e\n",
                        stgs->eps, stgs->alpha, (int)stgs->max_ipm_iters, (int)stgs->max_admm_iters,
                        (int)stgs->normalize, stgs->scale,  (int)stgs->adaptive, stgs->adaptive_lookback, stgs->rho_y);
    }
    else
    {
        abip_printf("eps = %.2e, alpha = %.2f, max_ipm_iters = %i, max_admm_iters = %i, normalize = %i\n"
                        "adaptive = %i, adaptive_lookback = %i, rho_y = %.2e\n",
                        stgs->eps, stgs->alpha, (int)stgs->max_ipm_iters, (int)stgs->max_admm_iters,
                        (int)stgs->normalize, (int)stgs->adaptive, stgs->adaptive_lookback, stgs->rho_y);
    }

    abip_printf("Variables n = %i, constraints m = %i\n", (int)d->n, (int)d->m);

#ifdef MATLAB_MEX_FILE

    mexEvalString("drawnow;");

#endif

    RETURN;
}
static void populate_on_failure
 (
  abip_int m,
  abip_int n,
  ABIPSolution *sol,
  ABIPInfo *info,
  abip_int status_val,
  const char *msg
 )
{
    DEBUG_FUNC

    if (info)
    {
        info->res_pri = NAN;
        info->res_dual = NAN;
        info->rel_gap = NAN;
        info->res_infeas = NAN;
        info->res_unbdd = NAN;

        info->pobj = NAN;
        info->dobj = NAN;

        info->ipm_iter = -1;
        info->admm_iter = -1;
        info->status_val = status_val;
        info->solve_time = NAN;
        strcpy(info->status, msg);
    }

    if (sol)
    {
        if (n > 0)
        {
            if (!sol->x)
            {
                sol->x = (abip_float *)abip_malloc(sizeof(abip_float) * n);
            }
            ABIP(scale_array)(sol->x, NAN, n);

            if (!sol->s)
            {
                sol->s = (abip_float *)abip_malloc(sizeof(abip_float) * n);
            }
            ABIP(scale_array)(sol->s, NAN, m);
        }
```

```
        if (m > 0)
        {
            if (!sol->y)
            {
                sol->y = (abip_float *)abip_malloc(sizeof(abip_float) * m);
            }
            ABIP(scale_array)(sol->y, NAN, m);
        }
    }

    RETURN;
}

static abip_int failure
 (
  ABIPWork *w,
  abip_int m,
  abip_int n,
  ABIPSolution *sol,
  ABIPInfo *info,
  abip_int stint,
  const char *msg,
  const char *ststr
  )
{
    DEBUG_FUNC

    abip_int status = stint;
    populate_on_failure(m, n, sol, info, status, ststr);

    abip_printf("Failure:%s\n", msg);
    abip_end_interrupt_listener();

    RETURN status;
}
static void warm_start_vars
 (
  ABIPWork *w,
  const ABIPSolution *sol
  )
{
    DEBUG_FUNC

    abip_int i;
    abip_int n = w->n;
    abip_int m = w->m;

    memset(w->v, 0, m * sizeof(abip_float));
    memcpy(w->u, sol->y, m * sizeof(abip_float));
    memcpy(&(w->u[m]), sol->x, n * sizeof(abip_float));
    memcpy(&(w->v[m]), sol->s, n * sizeof(abip_float));
    w->u[n + m] = 1.0;
    w->v[n + m] = 0.0;

#ifndef NOVALIDATE

    for (i = 0; i < n + m + 1; ++i)
    {
        if (abip_isnan(w->u[i]) && i<m)
        {
            w->u[i] = 0;
        }
        else
        {
            w->u[i] = SQRTF(w->mu/w->beta);
        }

        if (abip_isnan(w->v[i]))
        {
            w->v[i] = 0;
        }
        else
        {
            w->v[i] = SQRTF(w->mu/w->beta);
        }
    }

#endif

    if (w->stgs->normalize)
    {
        ABIP(normalize_warm_start)(w);
    }

    RETURN;
}
```

```
static void cold_start_vars
 (
  ABIPWork *w
  )
{
    DEBUG_FUNC

    abip_int l = w->m + w->n + 1;
    abip_int i;

    memset(w->u, 0, w->m * sizeof(abip_float));
    memset(w->v, 0, w->m * sizeof(abip_float));

    for (i = w->m; i < l; ++i)
    {
        w->u[i] = SQRTF(w->mu/w->beta);
        w->v[i] = SQRTF(w->mu/w->beta);
    }

    RETURN;
}
static abip_float calc_primal_resid
 (
  ABIPWork *w,
  const abip_float *x,
  const abip_float tau,
  abip_float *nm_A_x
  )
{
    DEBUG_FUNC

    abip_int i;

    abip_float pres = 0;
    abip_float scale;
    abip_float *pr = w->pr;

    *nm_A_x = 0;

    // compute w->pr again
    memset(pr, 0, w->m * sizeof(abip_float));
    ABIP(accum_by_A)(w->A, w->p, x, pr);

    for (i = 0; i < w->m; ++i)
    {
        scale = w->stgs->normalize ? w->scal->D[i] / (w->sc_b * w->stgs->scale) : 1;
        scale = scale * scale;
        *nm_A_x += (pr[i] * pr[i]) * scale;
        pres += (pr[i] - w->b[i] * tau) * (pr[i] - w->b[i] * tau) * scale;
    }

    *nm_A_x = SQRTF(*nm_A_x);
    RETURN SQRTF(pres);
}
static abip_float calc_dual_resid
 (
  ABIPWork *w,
  const abip_float *y,
  const abip_float *s,
  const abip_float tau,
  abip_float *nm_At_ys
  )
{
    DEBUG_FUNC

    abip_int i;
    abip_float dres = 0;
    abip_float scale;
    abip_float *dr = w->dr;

    *nm_At_ys = 0;

    memset(dr, 0, w->n * sizeof(abip_float));
    ABIP(accum_by_Atrans)(w->A, w->p, y, dr);
    ABIP(add_scaled_array)(dr, s, w->n, 1.0);

    for (i = 0; i < w->n; ++i)
    {
        scale = w->stgs->normalize ? w->scal->E[i] / (w->sc_c * w->stgs->scale) : 1;
        scale = scale * scale;
        *nm_At_ys += (dr[i] * dr[i]) * scale;
        dres += (dr[i] - w->c[i] * tau) * (dr[i] - w->c[i] * tau) * scale;
    }

    *nm_At_ys = SQRTF(*nm_At_ys);
    RETURN SQRTF(dres);
}
```

```
static void calc_residuals
 (
  ABIPWork *w,
  ABIPResiduals *r,
  abip_int ipm_iter,
  abip_int admm_iter
 )
{
    DEBUG_FUNC

    abip_float *y;
    abip_float *x;
    abip_float *s;

    // find y, x, s
    if (w->stgs->avg_criterion)
    {
        y = w->u_avgcon;
        x = &(w->u_avgcon[w->m]);
        s = &(w->v_avgcon[w->m]);
    }
    else
    {
        y = w->u;
        x = &(w->u[w->m]);
        s = &(w->v[w->m]);
    }

    abip_float nmpr_tau;
    abip_float nmdr_tau;
    abip_float nm_A_x_tau;
    abip_float nm_At_ys_tau;
    abip_float ct_x;
    abip_float bt_y;

    abip_int n = w->n;
    abip_int m = w->m;

    if (admm_iter && r->last_admm_iter == admm_iter)
    {
        RETURN;
    }

    r->last_ipm_iter = ipm_iter;
    r->last_admm_iter = admm_iter;


    if (w->stgs->avg_criterion)
    {
        r->tau = ABS(w->u_avgcon[n + m]);
        r->kap = ABS(w->v_avgcon[n + m]) / (w->stgs->normalize ? (w->stgs->scale * w->sc_c * w->sc_b)
 : 1);
    }
    else
    {
        r->tau = ABS(w->u[n + m]);
        r->kap = ABS(w->v[n + m]) / (w->stgs->normalize ? (w->stgs->scale * w->sc_c * w->sc_b) : 1);
    }


    // compute primal resid. and dual resid. memset pr and dr, then compute.
    nmpr_tau = calc_primal_resid(w, x, r->tau, &nm_A_x_tau);
    nmdr_tau = calc_dual_resid(w, y, s, r->tau, &nm_At_ys_tau);

    r->bt_y_by_tau = ABIP(dot)(y, w->b, m) / (w->stgs->normalize ? (w->stgs->scale * w->sc_c *
 w->sc_b) : 1);
    r->ct_x_by_tau = ABIP(dot)(x, w->c, n) / (w->stgs->normalize ? (w->stgs->scale * w->sc_c *
 w->sc_b) : 1);

    r->res_infeas = r->bt_y_by_tau > 0 ? w->nm_b * nm_At_ys_tau / r->bt_y_by_tau : NAN;
    r->res_unbdd = r->ct_x_by_tau < 0 ? w->nm_c * nm_A_x_tau / -r->ct_x_by_tau : NAN;

    bt_y = SAFEDIV_POS(r->bt_y_by_tau, r->tau);
    ct_x = SAFEDIV_POS(r->ct_x_by_tau, r->tau);

    r->res_pri = SAFEDIV_POS(nmpr_tau / (1 + w->nm_b), r->tau);
    r->res_dual = SAFEDIV_POS(nmdr_tau / (1 + w->nm_c), r->tau);
    r->rel_gap = ABS(ct_x - bt_y) / (1 + ABS(ct_x) + ABS(bt_y));

    RETURN;
}
static abip_int project_lin_sys
 (
  ABIPWork *w,
  abip_int iter
 )
```

```
{
    DEBUG_FUNC

    abip_int n = w->n;
    abip_int m = w->m;
    abip_int l = n + m + 1;

    abip_int status;
    memcpy(w->u_t, w->u, l * sizeof(abip_float));
    ABIP(add_scaled_array)(w->u_t, w->v, l, 1.0);

    ABIP(scale_array)(w->u_t, w->stgs->rho_y, m);
    ABIP(add_scaled_array)(w->u_t, w->h, l - 1, -w->u_t[l - 1]);
    ABIP(add_scaled_array)(w->u_t, w->h, l - 1, -ABIP(dot)(w->u_t, w->g, l - 1) / (w->g_th + 1));
    ABIP(scale_array)(&(w->u_t[m]), -1, n);
    status = ABIP(solve_lin_sys)(w->A, w->stgs, w->p, w->u_t, w->u, iter);
    w->u_t[l - 1] += ABIP(dot)(w->u_t, w->h, l - 1);
    RETURN status;
}

static void update_dual_vars
 (
  ABIPWork *w
  )
{
    DEBUG_FUNC

    abip_int i;
    abip_int m = w->m;
    abip_int l = m + w->n + 1;

    for (i = m; i < l; ++i)
    {
        w->v[i] += (w->u[i] - w->stgs->alpha * w->u_t[i] - (1.0 - w->stgs->alpha) * w->u_prev[i]);
    }

    RETURN;
}
/* @brief use restart strategy
*/
static void restart_vars
 (
  ABIPWork* w,
  abip_int admm_iter,
  abip_int total_admm_iter
  )

{
    DEBUG_FUNC

    abip_int i;
    abip_int fre = w->stgs->restart_fre;
    abip_int m = w->m;
    abip_int l = m + w->n + 1;

    for (i = 0; i < l; ++i)
    {
        w->u_avg[i] += w->u[i];
        w->v_avg[i] += w->v[i];
    }

    if (total_admm_iter < w->stgs->restart_thresh ||
        (admm_iter + 1 - w->fre_old ) % fre != 0)
    {
        return;
    }
    for (i = 0; i < l; ++i)
    {
        w->u_avg[i] /= fre;
        w->v_avg[i] /= fre;
    }

    memcpy(w->u, w->u_avg, sizeof(abip_float) * l);
    memcpy(w->v, w->v_avg, sizeof(abip_float) * l);
    memset(w->u_avg, 0, sizeof(abip_float) * l);
    memset(w->v_avg, 0, sizeof(abip_float) * l);

    //   memset(w->u_sumcon, 0, sizeof(abip_float) * l);
    //   memset(w->v_sumcon, 0, sizeof(abip_float) * l);

    w->fre_old = fre;


}

static void compute_avg
 (
```

```
 ABIPWork* w,
 abip_int admm_iter
 )
{
    DEBUG_FUNC

    abip_int i;
    abip_int m = w->m;
    abip_int l = m + w->n + 1;

    abip_int dom = admm_iter + 1;

    for (i = 0; i < l; ++i)
    {
        w->u_sumcon[i] += w->u[i] ;
        w->v_sumcon[i] += w->v[i] ;

        w->u_avgcon[i] = w->u_sumcon[i]/dom;
        w->v_avgcon[i] = w->v_sumcon[i]/dom;
    }


}


// add by Kurt. 22.04.11
static void half_update_dual_vars
 (
 ABIPWork *w
 )
{
    DEBUG_FUNC

    abip_int i;
    abip_int l = w->m + w->n + 1;

    for (i = 0; i < l; ++i)
    {
        w->v[i] += 0.5 * (w->u[i] - w->u_t[i] );
    }

    RETURN;
}

static void project_barrier_dual
 (
 ABIPWork *w
 )
{
    DEBUG_FUNC

    abip_int i;
    abip_int m = w->m;
    abip_int l = m + w->n + 1;
    abip_float tmp;
    // update u
    for (i = 0; i < l; ++i)
    {
        w->u[i] = w->u_t[i] - w->v[i];
    }

    for(i = m; i < l; ++i)
    {
        tmp = w->u[i] / 2;
        w->u[i] = tmp + SQRTF(tmp * tmp + w->mu / w->beta);
    }

    // dual update
    for (i = 0; i < l; ++i)
    {
        w->v[i] += (w->u[i] -  w->u_t[i]);
    }

    RETURN;
}


static void project_barrier
 (
 ABIPWork *w
 )
{
    DEBUG_FUNC

    abip_int i;
    abip_int m = w->m;
    abip_int l = m + w->n + 1;
```

```
    abip_int status;

    abip_float tmp;

    for (i = 0; i < m; ++i)
    {
        w->u[i] = w->u_t[i] - w->v[i];
    }

    for (i = m; i < l; ++i)
    {
        w->u[i] = w->stgs->alpha * w->u_t[i] + (1 - w->stgs->alpha) * w->u_prev[i] - w->v[i];
    }

    for(i = m; i < l; ++i)
    {
        tmp = w->u[i] / 2;
        w->u[i] = tmp + SQRTF(tmp * tmp + w->mu / w->beta);
    }

    RETURN;
}

static void update_barrier
 (
  ABIPWork* w,
  ABIPResiduals* r
  )
{
    abip_float sigma, gamma, mu = w->mu;

    abip_float ratio = w->mu / w->stgs->eps;
    abip_float err_ratio = MAX(MAX(r->res_pri, r->res_dual), r->rel_gap) / w->stgs->eps;

    if (MAX(w->sp, w->stgs->sparsity_ratio) > 0.4 || MIN(w->sp, w->stgs->sparsity_ratio) > 0.1)
    {
        if (ratio > 10.0)
        {
            gamma = 2.0;
        }
        else if (ratio > 1.0 && ratio <= 10.0)
        {
            gamma = 1.0;
        }
        else if (ratio > 0.5 && ratio <= 1.0)
        {
            gamma = 0.9;
        }
        else if (ratio > 0.1 && ratio <= 0.5)
        {
            gamma = 0.8;
        }
        else if (ratio > 0.05 && ratio <= 0.1)
        {
            gamma = 0.7;
        }
        else if (ratio > 0.01 && ratio <= 0.05)
        {
            gamma = 0.6;
        }
        else if (ratio > 0.005 && ratio <= 0.01)
        {
            gamma = 0.5;
        }
        else if (ratio > 0.001 && ratio <= 0.005)
        {
            gamma = 0.4;
        }
        else
        {
            gamma = 0.3;
        }

        if (err_ratio > 6 && err_ratio <= 10)
        {
            sigma = 0.5;
        }
        else if (err_ratio > 3 && err_ratio <= 6)
        {
            sigma = 0.6;
            gamma = gamma * 0.8;
        }
        else if (err_ratio > 1 && err_ratio <= 3)
        {
            w->final_check = 1;
            gamma = gamma * 0.4;
            if (ratio < 0.1)
```

```
            {
                sigma = 0.8;
            }
            else
            {
                sigma = 0.7;
            }

        }
        else
        {
            sigma = w->sigma;
        }
    }
    else
    {
        if (ratio > 10.0)
        {
            gamma = 3.0;
        }
        else if (ratio > 1.0 && ratio <= 10.0)
        {
            gamma = 1.0;
        }
        else if (ratio > 0.5 && ratio <= 1.0)
        {
            gamma = 0.9;
        }
        else if (ratio > 0.1 && ratio <= 0.5)
        {
            gamma = 0.8;
        }
        else if (ratio > 0.05 && ratio <= 0.1)
        {
            gamma = 0.7;
        }
        else if (ratio > 0.01 && ratio <= 0.05)
        {
            gamma = 0.6;
        }
        else if (ratio > 0.005 && ratio <= 0.01)
        {
            gamma = 0.5;
        }
        else if (ratio > 0.001 && ratio <= 0.005)
        {
            gamma = 0.4;
        }
        else
        {
            gamma = 0.3;
        }

        if (err_ratio > 6 && err_ratio <= 10)
        {
            sigma = 0.82;
            gamma = gamma * 0.8;
        }
        else if (err_ratio > 4 && err_ratio <= 6)
        {
            sigma = 0.84;
            gamma = gamma * 0.6;
        }
        else if (err_ratio > 3 && err_ratio <= 4)
        {
            sigma = 0.85;
            gamma = gamma * 0.5;
            w->final_check = 1;
        }
        else if (err_ratio > 1 && err_ratio <= 3)
        {
            w->final_check = 1;
            if (ratio < 0.1)
            {
                if (w->double_check)
                {
                    sigma = 0.9;
                    gamma = gamma * 0.4;
                    w->double_check = 0;
                }
                else
                {
                    sigma = 1.0;
                    gamma = gamma * 0.1;
                    w->double_check = 1;
                }
            }
```

```
            else
            {
                sigma = 0.88;
                gamma = gamma * 0.4;
            }
        }
        else
        {
            sigma = w->sigma;
        }
    }

    mu = mu * sigma;

    w->mu = mu;
    w->sigma = sigma;
    w->gamma = gamma;
}

// #ifndef MYDEBUG
// #define MYDEBUG
// #endif

static void update_barrier_dynamic
 (
  ABIPWork *w,
  ABIPResiduals *r
  )
{
    /*
     Implementation of a more delicate mu adjustment strategy

     ksi = min_i {x_i * z_i} / (x' * z / n)
     sigma = 0.1 * min(0.05 * (1 - ksi) / ksi, 2)^3

     */

    abip_int m = w->m, n = w->n, l = m + n + 1, i;

    double *u;
    double *v;
    double shrink = w->stgs->dynamic_sigma;

    if (w->stgs->avg_criterion){
        u = w->u_avgcon;
        v = w->v_avgcon;
    }
    else
    {
        u = w->u;
        v = w->v;
    }

    double ksi = 0.0, sigma = 0.0, xisi = 0.0, xs = 0.0, minxs = 1e+10;

    for (i = m; i < l; ++i) {
        xisi = u[i] * v[i];
        xs += xisi; minxs = MIN(xisi, minxs);
    }

    if (minxs <= 0.0) {
        abip_printf("Invalid xisi < 0 \n");
        assert(0);
    }

    xs /= (n + 1); ksi = minxs / xs;
    sigma = MIN(0.05 * (1 - ksi) / ksi, 2.0);
    sigma = MAX(0.1 * sigma * sigma * sigma, shrink);
    w->mu *= sigma;
    // w->gamma = gamma;
}

static void update_barrier_dynamic_2
 (
  ABIPWork* w

  ) {

    abip_float x = w->stgs->dynamic_x;
    abip_float eta = w->stgs->dynamic_sigma;
    w->mu *= MIN(x * w->mu, pow(w->mu, eta));
    return;
}
static void reinitialize_vars
 (
  ABIPWork *w,
  abip_int indx
```

```
  )
{
    DEBUG_FUNC

    abip_int i;
    abip_int m = w->m;
    abip_int l = m + w->n + 1;

    if (w->stgs->avg_criterion) // restart related
    {
        if (indx == 0)
        {
            for (i = m; i < l; ++i)
            {
                if (w-> u_avgcon[i] > w->v_avgcon[i])
                {
                    w->v_avgcon[i] = w->sigma * w->v_avgcon[i];
                }
                else
                {
                    w->u_avgcon[i] = w->sigma * w->u_avgcon[i];
                }
            }
        }
        else if (indx == 1)
        {
            for (i = m; i < l; ++i)
            {
                w->u_avgcon[i] = SQRTF(w->sigma) * w->u_avgcon[i];
                w->v_avgcon[i] = SQRTF(w->sigma) * w->v_avgcon[i];
            }
        }
        else
        {
            for (i = m; i < l; ++i)
            {
                w->u_avgcon[i] = SQRTF(1.0/w->sigma) * w->u_avgcon[i];
                w->v_avgcon[i] = SQRTF(1.0/w->sigma) * w->v_avgcon[i];
            }
        }
    }
    else
    {
        if (indx == 0)
        {
            for (i = m; i < l; ++i)
            {
                if (w-> u[i] > w->v[i])
                {
                    w->v[i] = w->sigma * w->v[i];
                }
                else
                {
                    w->u[i] = w->sigma * w->u[i];
                }
            }
        }
        else if (indx == 1)
        {
            for (i = m; i < l; ++i)
            {
                w->u[i] = SQRTF(w->sigma) * w->u[i];
                w->v[i] = SQRTF(w->sigma) * w->v[i];
            }
        }
        else
        {
            for (i = m; i < l; ++i)
            {
                w->u[i] = SQRTF(1.0/w->sigma) * w->u[i];
                w->v[i] = SQRTF(1.0/w->sigma) * w->v[i];
            }
        }
    }
    RETURN;
}
static abip_int indeterminate
 (
  ABIPWork *w,
  ABIPSolution *sol,
  ABIPInfo *info
  )
{
    DEBUG_FUNC

    strcpy(info->status, "Indeterminate");

```

```
    ABIP(scale_array)(sol->x, NAN, w->n);
    ABIP(scale_array)(sol->y, NAN, w->m);
    ABIP(scale_array)(sol->s, NAN, w->n);

    RETURN ABIP_INDETERMINATE;
}

static abip_int solved
 (
  ABIPWork *w,
  ABIPSolution *sol,
  ABIPInfo *info,
  abip_float tau
  )
{
    DEBUG_FUNC

    ABIP(scale_array)(sol->x, SAFEDIV_POS(1.0, tau), w->n);
    ABIP(scale_array)(sol->y, SAFEDIV_POS(1.0, tau), w->m);
    ABIP(scale_array)(sol->s, SAFEDIV_POS(1.0, tau), w->n);

    if (info->status_val == 0)
    {
        strcpy(info->status, "Solved/Inaccurate");
        RETURN ABIP_SOLVED_INACCURATE;
    }

    strcpy(info->status, "Solved");

    RETURN ABIP_SOLVED;
}
static void sety
 (
  ABIPWork *w,
  ABIPSolution *sol
  )
{
    DEBUG_FUNC

    if (!sol->y)
    {
        sol->y = (abip_float *) abip_malloc(sizeof(abip_float) * w->m);
    }

    if (w->stgs->avg_criterion)
    {
        memcpy(sol->y, w->u_avgcon, w->m * sizeof(abip_float));

    }
    else
    {
        memcpy(sol->y, w->u, w->m * sizeof(abip_float));

    }

    RETURN;
}
static void setx
 (
  ABIPWork *w,
  ABIPSolution *sol
  )
{
    DEBUG_FUNC

    if (!sol->x)
    {
        sol->x = (abip_float *) abip_malloc(sizeof(abip_float) * w->n);
    }

    if (w->stgs->avg_criterion)
    {
        memcpy(sol->x, &(w->u_avgcon[w->m]), w->n * sizeof(abip_float));
    }
    else
    {
        memcpy(sol->x, &(w->u[w->m]), w->n * sizeof(abip_float));

    }


    RETURN;
}
static void sets
 (
  ABIPWork *w,
  ABIPSolution *sol
```

```
 )
{
    DEBUG_FUNC

    if (!sol->s)
    {
        sol->s = (abip_float *) abip_malloc(sizeof(abip_float) * w->n);
    }

    if (w->stgs->avg_criterion)
    {
        memcpy(sol->s, &(w->v_avgcon[w->m]), w->n * sizeof(abip_float));
    }
    else
    {
        memcpy(sol->s, &(w->v[w->m]), w->n * sizeof(abip_float));

    }


    RETURN;
}
static abip_int infeasible
 (
 ABIPWork *w,
 ABIPSolution *sol,
 ABIPInfo *info,
 abip_float bt_y
 )
{
    DEBUG_FUNC

    ABIP(scale_array)(sol->y, 1 / bt_y, w->m);
    ABIP(scale_array)(sol->s, 1 / bt_y, w->n);
    ABIP(scale_array)(sol->x, NAN, w->n);

    if (info->status_val == 0)
    {
        strcpy(info->status, "Infeasible/Inaccurate");
        RETURN ABIP_INFEASIBLE_INACCURATE;
    }

    strcpy(info->status, "Infeasible");
    RETURN ABIP_INFEASIBLE;
}
static abip_int unbounded
 (
 ABIPWork *w,
 ABIPSolution *sol,
 ABIPInfo *info,
 abip_float ct_x
 )
{
    DEBUG_FUNC

    ABIP(scale_array)(sol->x, -1 / ct_x, w->n);
    ABIP(scale_array)(sol->y, NAN, w->m);
    ABIP(scale_array)(sol->s, NAN, w->n);

    if (info->status_val == 0)
    {
        strcpy(info->status, "Unbounded/Inaccurate");
        RETURN ABIP_UNBOUNDED_INACCURATE;
    }

    strcpy(info->status, "Unbounded");
    RETURN ABIP_UNBOUNDED;
}
static abip_int is_solved_status
 (
 abip_int status
 )
{
    RETURN status == ABIP_SOLVED || status == ABIP_SOLVED_INACCURATE;
}
static abip_int is_infeasible_status
 (
 abip_int status
 )
{
    RETURN status == ABIP_INFEASIBLE || status == ABIP_INFEASIBLE_INACCURATE;
}
static abip_int is_unbounded_status
 (
 abip_int status
 )
{
```

```
    RETURN status == ABIP_UNBOUNDED || status == ABIP_UNBOUNDED_INACCURATE;
}
static void get_info
 (
  ABIPWork *w,
  ABIPSolution *sol,
  ABIPInfo *info,
  ABIPResiduals *r,
  abip_int ipm_iter,
  abip_int admm_iter
  )
{
    DEBUG_FUNC

    info->ipm_iter = ipm_iter + 1;
    info->admm_iter = admm_iter + 1;

    info->res_infeas = r->res_infeas;
    info->res_unbdd = r->res_unbdd;

    if (is_solved_status(info->status_val))
    {
        info->rel_gap = r->rel_gap;
        info->res_pri = r->res_pri;
        info->res_dual = r->res_dual;
        info->pobj = r->ct_x_by_tau / r->tau;
        info->dobj = r->bt_y_by_tau / r->tau;
    }
    else if (is_unbounded_status(info->status_val))
    {
        info->rel_gap = NAN;
        info->res_pri = NAN;
        info->res_dual = NAN;
        info->pobj = -INFINITY;
        info->dobj = -INFINITY;
    }
    else if (is_infeasible_status(info->status_val))
    {
        info->rel_gap = NAN;
        info->res_pri = NAN;
        info->res_dual = NAN;
        info->pobj = INFINITY;
        info->dobj = INFINITY;
    }

    RETURN;
}
static void get_solution
 (
  ABIPWork *w,
  ABIPSolution *sol,
  ABIPInfo *info,
  ABIPResiduals *r,
  abip_int ipm_iter,
  abip_int admm_iter
  )
{
    DEBUG_FUNC

    abip_int l = w->m + w->n + 1;

    calc_residuals(w, r, ipm_iter, admm_iter);
    setx(w, sol);
    sety(w, sol);
    sets(w, sol);

    abip_float *tmp;

    if (w->stgs->avg_criterion)
    {
        tmp = w->u_avgcon;
    }
    else
    {
        tmp = w->u;
    }

    if (info->status_val == ABIP_UNFINISHED)
    {
        if (r->tau > INDETERMINATE_TOL && r->tau > r->kap)
        {
            info->status_val = solved(w, sol, info, r->tau);
        }
        else if (ABIP(norm)(tmp, l) < INDETERMINATE_TOL * SQRTF((abip_float)l))
        {
            info->status_val = indeterminate(w, sol, info);
        }
```

```
        else if (-r->bt_y_by_tau < r->ct_x_by_tau)
        {
            info->status_val = infeasible(w, sol, info, r->bt_y_by_tau);
        }
        else
        {
            info->status_val = unbounded(w, sol, info, r->ct_x_by_tau);
        }
    }
    else if (is_solved_status(info->status_val))
    {
        info->status_val = solved(w, sol, info, r->tau);
    }
    else if (is_infeasible_status(info->status_val))
    {
        info->status_val = infeasible(w, sol, info, r->bt_y_by_tau);
    }
    else
    {
        info->status_val = unbounded(w, sol, info, r->ct_x_by_tau);
    }

    if (w->stgs->normalize)
    {
        ABIP(un_normalize_sol)(w, sol);
    }

    get_info(w, sol, info, r, ipm_iter, admm_iter);

    RETURN;
}
static void print_summary
 (
  ABIPWork *w,
  abip_int i,
  abip_int j,
  ABIPResiduals *r,
  ABIP(timer) *solve_timer
  )
{
    DEBUG_FUNC

    abip_printf("%*i|", (int) strlen(HEADER[0]), (int) i);
    abip_printf("%*i|", (int) strlen(HEADER[1]), (int) j);
    abip_printf("%*.2e|", (int) strlen(HEADER[2]), w->mu);

    abip_printf("%*.2e|", (int) HSPACE, r->res_pri);
    abip_printf("%*.2e|", (int) HSPACE, r->res_dual);
    abip_printf("%*.2e|", (int) HSPACE, r->rel_gap);
    abip_printf("%*.2e|", (int) HSPACE, SAFEDIV_POS(r->ct_x_by_tau, r->tau));
    abip_printf("%*.2e|", (int) HSPACE, SAFEDIV_POS(r->bt_y_by_tau, r->tau));
    abip_printf("%*.2e|", (int) HSPACE, SAFEDIV_POS(r->kap, r->tau));
    abip_printf("%*.2e ", (int) HSPACE, ABIP(tocq)(solve_timer) / 1e3);
    abip_printf("\n");

#if EXTRA_VERBOSE > 0

    abip_printf("Norm u = %4f, ", ABIP(norm)(w->u, w->n + w->m + 1));
    abip_printf("Norm u_t = %4f, ", ABIP(norm)(w->u_t, w->n + w->m + 1));
    abip_printf("Norm v = %4f, ", ABIP(norm)(w->v, w->n + w->m + 1));
    abip_printf("tau = %4f, ", r->tau);
    abip_printf("kappa = %4f, ", r->kap);
    abip_printf("|u - u_prev| = %1.2e, ", ABIP(norm_diff)(w->u, w->u_prev, w->n + w->m + 1));
    abip_printf("|u - u_t| = %1.2e, ", ABIP(norm_diff)(w->u, w->u_t, w->n + w->m + 1));
    abip_printf("res_infeas = %1.2e, ", r->res_infeas);
    abip_printf("res_unbdd = %1.2e\n", r->res_unbdd);

#endif

#ifdef MATLAB_MEX_FILE

    mexEvalString("drawnow;");

#endif

    RETURN;
}
static void print_header
 (
  ABIPWork *w
  )
{
    DEBUG_FUNC

    abip_int i;

    if (w->stgs->warm_start)
```

```
    {
        abip_printf("ABIP using variable warm-starting\n");
    }

    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("-");
    }
    abip_printf("\n");

    for (i = 0; i < HEADER_LEN - 1; ++i)
    {
        abip_printf("%s|", HEADER[i]);
    }
    abip_printf("%s\n", HEADER[HEADER_LEN - 1]);

    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("-");
    }
    abip_printf("\n");

#ifdef MATLAB_MEX_FILE

    mexEvalString("drawnow;");

#endif

    RETURN;
}
static void print_footer
 (
  const ABIPData *d,
  ABIPSolution *sol,
  ABIPWork *w,
  ABIPInfo *info
  )
{
    DEBUG_FUNC

    abip_int i;

    char *lin_sys_str = ABIP(get_lin_sys_summary)(w->p, info);
    char *adapt_str = ABIP(get_adapt_summary)(info, w->adapt);

    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("-");
    }

    abip_printf("\n");

    abip_printf("Status: %s\n", info->status);

    if (info->ipm_iter+1 == w->stgs->max_ipm_iters)
    {
        abip_printf("Hit max_ipm_iters, solution may be inaccurate\n");
    }

    if (info->admm_iter+1 >= w->stgs->max_admm_iters)
    {
        abip_printf("Hit max_admm_iters, solution may be inaccurate\n");
    }

    abip_printf("Timing: Solve time: %1.2es\n", info->solve_time / 1e3);

    if (lin_sys_str)
    {
        abip_printf("%s", lin_sys_str);
        abip_free(lin_sys_str);
    }

    if (adapt_str)
    {
        abip_printf("%s", adapt_str);
        abip_free(adapt_str);
    }

    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("-");
    }

    abip_printf("\n");

    if (is_infeasible_status(info->status_val))
    {
```

```
        abip_printf("Certificate of primal infeasibility:\n");
        abip_printf("|A'y + s|_2 * |b|_2 = %.4e\n", info->res_infeas);
        abip_printf("b'y = %.4f\n", ABIP(dot)(d->b, sol->y, d->m));
    }
    else if (is_unbounded_status(info->status_val))
    {
        abip_printf("Certificate of dual infeasibility:\n");
        abip_printf("|Ax|_2 * |c|_2 = %.4e\n", info->res_unbdd);
        abip_printf("c'x = %.4f\n", ABIP(dot)(d->c, sol->x, d->n));
    }
    else
    {
        abip_printf("Error metrics:\n");
        abip_printf("primal res: |Ax - b|_2 / (1 + |b|_2) = %.4e\n", info->res_pri);
        abip_printf("dual res: |A'y + s - c|_2 / (1 + |c|_2) = %.4e\n", info->res_dual);
        abip_printf("rel gap: |c'x - b'y| / (1 + |c'x| + |b'y|) = %.4e\n", info->rel_gap);

        for (i = 0; i < LINE_LEN; ++i)
        {
            abip_printf("-");
        }

        abip_printf("\n");
        abip_printf("c'x = %.4e, b'y = %.4e\n", info->pobj, info->dobj);
    }

    for (i = 0; i < LINE_LEN; ++i)
    {
        abip_printf("=");
    }

    abip_printf("\n");

#ifdef MATLAB_MEX_FILE

    mexEvalString("drawnow;");

#endif

    RETURN;
}

static abip_int has_converged
 (
  ABIPWork *w,
  ABIPResiduals *r,
  abip_int ipm_iter,
  abip_int admm_iter
  )
{
    DEBUG_FUNC

    abip_float eps = w->stgs->eps;

    if (r->res_pri < eps && (r->res_dual < eps || w->stgs->pfeasopt) && r->rel_gap < eps)
    {
        RETURN ABIP_SOLVED;
    }

    if (r->res_unbdd < eps && ipm_iter > 0 && admm_iter > 0)
    {
        RETURN ABIP_UNBOUNDED;
    }

    if (r->res_infeas < eps && ipm_iter > 0 && admm_iter > 0)
    {
        RETURN ABIP_INFEASIBLE;
    }

    RETURN 0;
}

static abip_int validate
 (
  const ABIPData *d
  )
{
    DEBUG_FUNC

    ABIPSettings *stgs = d->stgs;

    if (d->m <= 0 || d->n <= 0)
    {
        abip_printf("m and n must both be greater than 0; m = %li, n = %li\n", (long) d->m, (long)
 d->n);
        RETURN - 1;
    }
```

```
    if (d->m > d->n)
    {
        abip_printf("WARN: m larger than n, problem likely degenerate\n");
        RETURN - 1;
    }

    if (ABIP(validate_lin_sys)(d->A) < 0)
    {
        abip_printf("invalid linear system input data\n");
        RETURN - 1;
    }

    if (stgs->max_ipm_iters <= 0)
    {
        abip_printf("max_ipm_iters must be positive\n");
        RETURN - 1;
    }

    if (stgs->max_admm_iters <= 0)
    {
        abip_printf("max_admm_iters must be positive\n");
        RETURN - 1;
    }

    if (stgs->eps <= 0)
    {
        abip_printf("eps tolerance must be positive\n");
        RETURN - 1;
    }

    if (stgs->alpha <= 0 || stgs->alpha >= 2)
    {
        abip_printf("alpha must be in (0,2)\n");
        RETURN - 1;
    }

    if (stgs->rho_y <= 0)
    {
        abip_printf("rho_y must be positive (1e-3 works well).\n");
        RETURN - 1;
    }

    if (stgs->scale <= 0)
    {
        abip_printf("scale must be positive (1 works well).\n");
        RETURN - 1;
    }

    if (stgs->eps_cor <= 0)
    {
        abip_printf("eps_cor tolerance must be positive.\n");
        RETURN - 1;
    }

    if (stgs->eps_pen <= 0)
    {
        abip_printf("eps_pen tolerance must be positive.\n");
        RETURN - 1;
    }

    if (stgs->adaptive_lookback <= 0)
    {
        abip_printf("adaptive_lookback must be positive.\n");
        RETURN - 1;
    }

    if(stgs->hybrid_mu > 0 && stgs->dynamic_sigma >= 0 )
    {
        abip_printf("when use hybrid mu strategy, dynamic_sigma must be negative.\n");
        RETURN - 1;
    }

    RETURN 0;
}

static ABIPWork *init_work
 (
  const ABIPData *d
  )
{
    DEBUG_FUNC

    ABIPWork *w = (ABIPWork *) abip_calloc(1, sizeof(ABIPWork));
    abip_int l = d->n + d->m + 1;

    if (d->stgs->verbose)
```

```
    {
        print_init_header(d);
    }

    if (!w)
    {
        abip_printf("ERROR: allocating work failure\n");
        RETURN ABIP_NULL;
    }

    w->stgs = d->stgs;
    w->m = d->m;
    w->n = d->n;

    w->u = (abip_float *) abip_malloc(l * sizeof(abip_float));
    w->v = (abip_float *) abip_malloc(l * sizeof(abip_float));
    w->u_t = (abip_float *) abip_malloc(l * sizeof(abip_float));
    w->u_prev = (abip_float *) abip_malloc(l * sizeof(abip_float));
    w->v_prev = (abip_float *) abip_malloc(l * sizeof(abip_float));
    w->u_avg = (abip_float*) abip_malloc(l * sizeof(abip_float));
    w->v_avg = (abip_float*)abip_malloc(l * sizeof(abip_float));

    w->u_avgcon = (abip_float*) abip_malloc(l * sizeof(abip_float));
    w->v_avgcon = (abip_float*)abip_malloc(l * sizeof(abip_float));
    w->u_sumcon = (abip_float*) abip_malloc(l * sizeof(abip_float));
    w->v_sumcon = (abip_float*)abip_malloc(l * sizeof(abip_float));


    w->h = (abip_float *) abip_malloc((l - 1) * sizeof(abip_float));
    w->g = (abip_float *) abip_malloc((l - 1) * sizeof(abip_float));
    w->pr = (abip_float *) abip_malloc(d->m * sizeof(abip_float));
    w->dr = (abip_float *) abip_malloc(d->n * sizeof(abip_float));
    w->b = (abip_float *) abip_malloc(d->m * sizeof(abip_float));
    w->c = (abip_float *) abip_malloc(d->n * sizeof(abip_float));

    if (!w->u || !w->v || !w->u_t || !w->u_prev || !w->h || !w->g || !w->pr || !w->dr || !w->b || !w->c)
    {
        abip_free(w->u); abip_free(w->v); abip_free(w->u_t); abip_free(w->u_prev);
        abip_free(w->h); abip_free(w->g); abip_free(w->pr); abip_free(w->dr);
        abip_free(w->b); abip_free(w->c);
        abip_printf("ERROR: work memory allocation failure\n");
        RETURN ABIP_NULL;
    }

    w->A = d->A;
    w->sp = d->sp;

    if (w->stgs->normalize)
    {
#ifdef COPYAMATRIX

        if (!ABIP(copy_A_matrix)(&(w->A), d->A))
        {
            abip_printf("ERROR: copy A matrix failed\n");
            RETURN ABIP_NULL;
        }

#endif

        w->scal = (ABIPScaling *)abip_malloc(sizeof(ABIPScaling));
        ABIP(normalize_A)(w->A, w->stgs, w->scal);

#if EXTRA_VERBOSE > 0

        ABIP(print_array)(w->scal->D, d->m, "D");
        abip_printf("ABIP(norm) D = %4f\n", ABIP(norm)(w->scal->D, d->m));
        ABIP(print_array)(w->scal->E, d->n, "E");
        abip_printf("ABIP(norm) E = %4f\n", ABIP(norm)(w->scal->E, d->n));

#endif
    }
    else
    {
        w->scal = ABIP_NULL;
    }

    if (!(w->p = ABIP(init_lin_sys_work)(w->A, w->stgs)))
    {
        abip_printf("ERROR: init_lin_sys_work failure\n");
        RETURN ABIP_NULL;
    }

    if (!(w->adapt = ABIP(init_adapt)(w)))
    {
        abip_printf("ERROR: init_adapt failure\n");
        RETURN ABIP_NULL;
```

```
    }

    RETURN w;
}
static abip_int update_work
 (
  const ABIPData *d,
  ABIPWork *w,
  const ABIPSolution *sol
  )
{
    DEBUG_FUNC

    abip_int n = d->n;
    abip_int m = d->m;

    w->nm_b = ABIP(norm)(d->b, m);
    w->nm_c = ABIP(norm)(d->c, n);
    memcpy(w->b, d->b, d->m * sizeof(abip_float));
    memcpy(w->c, d->c, d->n * sizeof(abip_float));

#if EXTRA_VERBOSE > 0

    ABIP(print_array)(w->b, m, "b");
    abip_printf("pre-normalized norm b = %4f\n", ABIP(norm)(w->b, m));
    ABIP(print_array)(w->c, n, "c");
    abip_printf("pre-normalized norm c = %4f\n", ABIP(norm)(w->c, n));

#endif

    if (w->stgs->normalize)
    {
        ABIP(normalize_b_c)(w);

#if EXTRA_VERBOSE > 0

        ABIP(print_array)(w->b, m, "bn");
        abip_printf("sc_b = %4f\n", w->sc_b);
        abip_printf("post-normalized norm b = %4f\n", ABIP(norm)(w->b, m));

        ABIP(print_array)(w->c, n, "cn");
        abip_printf("sc_c = %4f\n", w->sc_c);
        abip_printf("post-normalized norm c = %4f\n", ABIP(norm)(w->c, n));

#endif
    }

    if (MAX(w->sp,w->stgs->sparsity_ratio) > 0.4 || (MIN(w->sp,w->stgs->sparsity_ratio) > 0.1 && MIN(w->sp,w->stgs->sparsity_ratio) < 0.2))
    {
        w->sigma = 0.3;
        w->gamma = 2.0;
    }
    else if (MIN(w->sp,w->stgs->sparsity_ratio) > 0.2)
    {
        w->sigma = 0.5;
        w->gamma = 3.0;
    }
    else
    {
        w->sigma = 0.8;
        w->gamma = 3.0;
    }

    w->final_check = 0;
    w->double_check = 0;

    w->mu = 1.0;
    w->beta = 1.0;

    if (w->stgs->warm_start)
    {
        warm_start_vars(w, sol);
    }
    else
    {
        cold_start_vars(w);
    }

    memcpy(w->h, w->b, m * sizeof(abip_float));
    memcpy(&(w->h[m]), w->c, n * sizeof(abip_float));
    ABIP(scale_array)(w->h, -1, m);
    memcpy(w->g, w->h, (n + m) * sizeof(abip_float));

    ABIP(solve_lin_sys)(w->A, w->stgs, w->p, w->g, ABIP_NULL, -1); // solve the linear system
    ABIP(scale_array)(&(w->g[m]), -1, n);
    w->g_th = ABIP(dot)(w->h, w->g, n + m);
```

```
    RETURN 0;
}

static abip_float iterate_norm_diff
 (
  ABIPWork *w
  )
{
    DEBUG_FUNC

    abip_int l = w->m + w->n + 1;

    abip_float u_norm_difference = ABIP(norm_diff)(w->u, w->u_prev, l);
    abip_float v_norm_difference = ABIP(norm_diff)(w->v, w->v_prev, l);
    abip_float norm = 1 + SQRTF(ABIP(norm_sq)(w->u, l) + ABIP(norm_sq)(w->v, l)) +
 SQRTF(ABIP(norm_sq)(w->u_prev, l) + ABIP(norm_sq)(w->v_prev, l));
    abip_float norm_diff = SQRTF(u_norm_difference * u_norm_difference + v_norm_difference *
 v_norm_difference);

    RETURN norm_diff / norm;
}
static abip_float iterate_Q_norm_resd
 (
  ABIPWork *w,
  abip_int j
  )
{
    DEBUG_FUNC

    abip_int i;
    abip_int l = w->m + w->n + 1;

    abip_float *y = w->u;
    abip_float *x = &(w->u[w->m]);
    abip_float *s = &(w->v[w->m]);
    abip_float tau = w->u[w->m + w->n];
    abip_float kap = w->v[w->m + w->n];

    abip_float Qres = 0;
    abip_float Qres_avg = w->stgs->max_admm_iters;
    abip_float norm_avg = 1;

    // initialize pr and dr
    abip_float *pr = w->pr;
    abip_float *dr = w->dr;

    memset(pr, 0, w->m * sizeof(abip_float));
    memset(dr, 0, w->n * sizeof(abip_float));

    // compute pr and dr
    ABIP(accum_by_A)(w->A, w->p, x, pr);
    ABIP(accum_by_Atrans)(w->A, w->p, y, dr);
    ABIP(add_scaled_array)(dr, s, w->n, 1.0);

    for (i = 0; i < w->m; ++i)
    {
        Qres += (pr[i] - w->b[i] * tau) * (pr[i] - w->b[i] * tau);
    }

    for (i = 0; i < w->n; ++i)
    {
        Qres += (dr[i] - w->c[i] * tau) * (dr[i] - w->c[i] * tau);
    }
    abip_float cTx = ABIP(dot)(x, w->c, w->n);
    abip_float bTy = ABIP(dot)(y, w->b, w->m);
    Qres += (bTy - cTx - kap) * (bTy - cTx - kap);
    abip_float norm = 1 + SQRTF(ABIP(norm_sq)(w->u, l) + ABIP(norm_sq)(w->v, l)) ;


    // check inner loop termination criterion via the average solution returned by restart strategy
    if ( (j+1) % 10 == 0)
    {
        // initialize pr_avg and dr_avg
        Qres_avg = 0;
        norm_avg = 0;
        abip_float *y_avg = w->u_avgcon;
        abip_float *x_avg = &(w->u_avgcon[w->m]);
        abip_float *s_avg = &(w->v_avgcon[w->m]);
        abip_float tau_avg = w->u_avgcon[w->m + w->n];
        abip_float kap_avg = w->v_avgcon[w->m + w->n];
        abip_float *pr_avg;
        abip_float *dr_avg;
        pr_avg = (abip_float *) abip_malloc(w->m * sizeof(abip_float));
        dr_avg = (abip_float *) abip_malloc(w->n * sizeof(abip_float));
        memset(pr_avg, 0, w->m * sizeof(abip_float));
        memset(dr_avg, 0, w->n * sizeof(abip_float));
```

```
        // compute pr_avg and dr_avg
        ABIP(accum_by_A)(w->A, w->p, x_avg, pr_avg);
        ABIP(accum_by_Atrans)(w->A, w->p, y_avg, dr_avg);
        ABIP(add_scaled_array)(dr_avg, s_avg, w->n, 1.0);
        for (i = 0; i < w->m; ++i)
        {
            Qres_avg += (pr_avg[i] - w->b[i] * tau_avg) * (pr_avg[i] - w->b[i] * tau_avg);
        }

        for (i = 0; i < w->n; ++i)
        {
            Qres_avg += (dr_avg[i] - w->c[i] * tau_avg) * (dr_avg[i] - w->c[i] * tau_avg);
        }

        abip_float cTx_avg = ABIP(dot)(x_avg, w->c, w->n);
        abip_float bTy_avg = ABIP(dot)(y_avg, w->b, w->m);

        Qres_avg += (bTy_avg - cTx_avg - kap_avg) * (bTy_avg - cTx_avg - kap_avg);

        norm_avg = 1 + SQRTF(ABIP(norm_sq)(w->u_avgcon, l) + ABIP(norm_sq)(w->v_avgcon, l));
        abip_free(pr_avg); abip_free(dr_avg);
    }

    if (SQRTF(Qres_avg) / norm_avg < SQRTF(Qres) / norm )
    {
        w->stgs->avg_criterion = 1;

        RETURN SQRTF(Qres_avg) / norm_avg;
    }
    else
    {
        w->stgs->avg_criterion = 0;
        RETURN SQRTF(Qres) / norm;
    }
}

abip_int ABIP(solve)
(
 ABIPWork *w,
 const ABIPData *d,
 ABIPSolution *sol,
 ABIPInfo *info
 )
{
    DEBUG_FUNC

    abip_int i;
    abip_int j;
    abip_int k;
    abip_int ii;
    abip_int inner_stopper;
    ABIP(timer) solve_timer;

    abip_int jj;

    ABIPResiduals r;
    abip_int l = w->m + w->n + 1;

    if (!d || !sol || !info || !w || !d->b || !d->c)
    {
        abip_printf("ERROR: ABIP_NULL input\n");
        RETURN ABIP_FAILED;
    }

    clock_t start_time = clock();
    double elapsedT = 0.0, maxTime = w->stgs->max_time;

    abip_start_interrupt_listener();
    ABIP(tic)(&solve_timer);

    info->status_val = ABIP_UNFINISHED;
    r.last_ipm_iter = -1;
    r.last_admm_iter = -1;
    update_work(d, w, sol);

    if (w->stgs->verbose)
    {
        print_header(w);
    }

    k = 0;

    for (i = 0; i < w->stgs->max_ipm_iters; ++i) // the outer loop
    {
        if (MIN(w->sp, w->stgs->sparsity_ratio) > 0.5) // determine the # iteratin of inner loop
        {
```

```
            inner_stopper = (int)round(POWF(w->mu, -0.35));
        }
        else if (MIN(w->sp, w->stgs->sparsity_ratio) > 0.2)
        {
            inner_stopper = (int)round(POWF(w->mu, -1));
        }
        else
        {
            inner_stopper = w->stgs->max_admm_iters;
        }
        w->fre_old = 0;
        memset(w->u_avg, 0, sizeof(abip_float) * l);
        memset(w->v_avg, 0, sizeof(abip_float) * l);


        memset(w->u_sumcon, 0, sizeof(abip_float) * l);
        memset(w->v_sumcon, 0, sizeof(abip_float) * l);


        if (w->stgs->avg_criterion)
        {
            memcpy(w->u, w->u_avgcon, sizeof(abip_float) * l);
            memcpy(w->v, w->v_avgcon, sizeof(abip_float) * l);
        }

        for (j = 0; j < inner_stopper; ++j) // the inner loop
        {
            memcpy(w->u_prev, w->u, l * sizeof(abip_float));
            memcpy(w->v_prev, w->v, l * sizeof(abip_float));

            // update variables
            if (project_lin_sys(w, k) < 0)
            {
                RETURN failure(w, w->m, w->n, sol, info, ABIP_FAILED, "error in project_lin_sys", "Failure");
            }


            if (w->stgs->half_update) // half update
            {
                half_update_dual_vars(w);

                project_barrier_dual(w);

            }

            else // normal update
            {
                project_barrier(w);

                update_dual_vars(w);

            }
            // restart
            restart_vars(w, j, k);


            if (abip_is_interrupted())
            {
                RETURN failure(w, w->m, w->n, sol, info, ABIP_SIGINT, "Interrupted", "Interrupted");
            }

            // compute average solution
            compute_avg(w,j);


            k += 1;

            if (iterate_Q_norm_resd(w, j) < w->gamma*w->mu) // inner loop termination criterion
            {
                if (w->stgs->half_update)
                {
                    for (jj=0; jj<l; ++jj)
                    {
                        if( w->v[jj] < 0 )
                        {
                            w->v[jj] = 1e-6;
                            // abip_printf("find a negative element\n");
                        }
                    }
                }

                break;
            }
            // check whether the inner loop iterate has satisfied the tolerance level
            if (w-> final_check && (j+1) % CONVERGED_INTERVAL == 0)
            {
```

```
                calc_residuals(w, &r, i, k); // compute residuals

                if ((info->status_val = has_converged(w, &r, i, k)) != 0 || k+1 >=
w->stgs->max_admm_iters || i+1 >= w->stgs->max_ipm_iters)
                {
                    if (w->stgs->verbose && k>0)
                    {
                        print_summary(w, i, k, &r, &solve_timer); // converge
                    }

                    get_solution(w, sol, info, &r, i, k);
                    info->solve_time = ABIP(tocq)(&solve_timer);

                    if (w->stgs->verbose)
                    {
                        print_footer(d, sol, w, info);
                    }

                    abip_end_interrupt_listener();

                    RETURN info->status_val;
                }
            }

        }

        elapsedT = ((abip_float)clock() - start_time) / CLOCKS_PER_SEC; // record runtime
        if (elapsedT > maxTime) {
            abip_printf("Timelimit reached. \n");
            w->stgs->max_admm_iters = k * 1.05;
        }

        // early stopping strategy
        if (w->mu < w->stgs->eps)
        {
            w->final_check = 1;
        }
        // check whether the iterate satisfies the tolerance level in the outer loop
        calc_residuals(w, &r, i, k);
        if (w->stgs->verbose)
        {
            print_summary(w, i, k, &r, &solve_timer);
        }

        if ((info->status_val = has_converged(w, &r, i, k)) != 0 || k+1 >= w->stgs->max_admm_iters)
        {
            get_solution(w, sol, info, &r, i, k);
            info->solve_time = ABIP(tocq)(&solve_timer);

            if (w->stgs->verbose)
            {
                print_footer(d, sol, w, info);
            }

            abip_end_interrupt_listener();

            RETURN info->status_val;
        }

        // update mu
        if (w->stgs->hybrid_mu)
        {
            if (w->stgs->dynamic_sigma_second > 0.0 && w->mu < w->stgs->hybrid_thresh * w->stgs->eps
){
                w->stgs->dynamic_sigma = w->stgs->dynamic_sigma_second;
                update_barrier_dynamic(w, &r);

            }
            else if (w->stgs->dynamic_sigma_second == 0.0 && w->mu < w->stgs->hybrid_thresh *
w->stgs->eps){
                w->stgs->dynamic_sigma = w->stgs->dynamic_sigma_second;
                update_barrier(w, &r);
            }
            else if (w->stgs->dynamic_sigma < 0.0){
                update_barrier_dynamic_2(w);
            }
        }
        else
        {
            if (w->stgs->dynamic_sigma == 0.0) {
                update_barrier(w, &r);
            }
            else if (w->stgs->dynamic_sigma < 0.0) {
                update_barrier_dynamic_2(w);
            }
            else {
                update_barrier_dynamic(w, &r);
```

```
            }
        }
        // prepare the next outer loop
        reinitialize_vars(w, 0);

        if (w->stgs->adaptive)
        {
            reinitialize_vars(w, 1);

            w->beta = 1;

            if (ABIP(adaptive)(w, k) < 0)
            {
                RETURN failure(w, w->m, w->n, sol, info, ABIP_FAILED, "error in adaptive", "Failure");
            }

            reinitialize_vars(w, 2);
        }
    }

    RETURN info->status_val; // return status
}

/* @brief recover the optimal solution and set memory free
*/
void ABIP(finish)
(
 ABIPWork *w
 )
{
    DEBUG_FUNC

    if (w)
    {
        if (w->stgs && w->stgs->normalize)
        {
#ifndef COPYAMATRIX
            ABIP(un_normalize_A)(w->A, w->stgs, w->scal);
#else
            ABIP(free_A_matrix)(w->A);
#endif
        }

        if (w->p)
        {
#ifdef ABIP_PARDISO
            ABIP(free_lin_sys_work_pds)(w->p, w->A);
#else
            ABIP(free_lin_sys_work)(w->p);
#endif
        }

        if (w->adapt)
        {
            ABIP(free_adapt)(w->adapt);
        }

        free_work(w);
    }

    RETURN;
}

/* @brief initialization
*/
ABIPWork *ABIP(init)
(
 const ABIPData *d,
 ABIPInfo *info
 )
{
    DEBUG_FUNC

#if EXTRA_VERBOSE > 1
    ABIP(tic)(&global_timer);
#endif

    ABIPWork *w;
    ABIP(timer) init_timer;
    abip_start_interrupt_listener();

    if (!d || !info)
    {
        abip_printf("ERROR: Missing ABIPData or ABIPInfo input\n");
        RETURN ABIP_NULL;
    }

```

```
#if EXTRA_VERBOSE > 0
    ABIP(print_data)(d);
#endif

#ifndef NOVALIDATE
    if (validate(d) < 0)
    {
        abip_printf("ERROR: Validation returned failure\n");
        RETURN ABIP_NULL;
    }
#endif

    ABIP(tic)(&init_timer);

    w = init_work(d); // initialization
    info->setup_time = ABIP(tocq)(&init_timer);

    if (d->stgs->verbose)
    {
        abip_printf("Setup time: %1.2es\n", info->setup_time / 1e3); // prinf information
    }

    abip_end_interrupt_listener();

    RETURN w;
}

abip_int ABIP(main)
(
 const ABIPData *d,
 ABIPSolution *sol,
 ABIPInfo *info
 )
{
    DEBUG_FUNC

    abip_int status;
    ABIPWork *w = ABIP(init)(d, info);

#if EXTRA_VERBOSE > 0
    abip_printf("size of abip_int = %lu, size of abip_float = %lu\n", sizeof(abip_int),
 sizeof(abip_float));
#endif

    if (w)
    {
        ABIP(solve)(w, d, sol, info); // solve the problem via abip
        status = info->status_val;
    }
    else
    {
        status = failure(ABIP_NULL, d ? d->m : -1, d ? d->n : -1, sol, info, ABIP_FAILED, "could not
 initialize work", "Failure");
    }

    ABIP(finish)(w);

    RETURN status;
}
```

5.94 src/abip_version.c File Reference

```
#include "glbopts.h"
```

Functions

- const char *ABIP() version (void)

    return the abip version

5.94.1 Function Documentation

5.94.1.1 version()

```
const char *ABIP() version (
            void )
```

return the abip version

Definition at line 5 of file abip_version.c.

5.95 abip_version.c

Go to the documentation of this file.

```
00001 #include "glbopts.h"
00005 const char *ABIP(version)(void)
00006 {
00007     return ABIP_VERSION;
00008 }
```

5.96 src/adaptive.c File Reference

```
#include "adaptive.h"
#include "linalg.h"
#include "linsys.h"
#include "abip.h"
#include "abip_blas.h"
#include "util.h"
```

Data Structures

- struct ABIP_ADAPTIVE_WORK

Functions

- ABIPAdaptWork *ABIP() init_adapt (ABIPWork *w)
- abip_int ABIP() adaptive (ABIPWork *w, abip_int iter)
- void ABIP() free_adapt (ABIPAdaptWork *a)
- char *ABIP() get_adapt_summary (const ABIPInfo *info, ABIPAdaptWork *a)

5.96.1 Function Documentation

5.96.1.1 adaptive()

```
abip_int ABIP() adaptive (
            ABIPWork * w,
            abip_int iter )
```

Definition at line 305 of file adaptive.c.

5.96.1.2 free_adapt()

```
void ABIP() free_adapt (
             ABIPAdaptWork * a )
```

Definition at line 336 of file adaptive.c.

5.96.1.3 get_adapt_summary()

```
char *ABIP() get_adapt_summary (
             const ABIPInfo * info,
             ABIPAdaptWork * a )
```

Definition at line 406 of file adaptive.c.

5.96.1.4 init_adapt()

```
ABIPAdaptWork *ABIP() init_adapt (
             ABIPWork * w )
```

Definition at line 258 of file adaptive.c.

5.97 adaptive.c

Go to the documentation of this file.

```
#include "adaptive.h"
#include "linalg.h"
#include "linsys.h"
#include "abip.h"
#include "abip_blas.h"
#include "util.h"

/* This file uses adaption to improve the convergence rate of the ADMM in each inner loop.
 * At each iteration we need to select a nearly-optimal penalty parameter beta, we do this using Barzilai-Borwein spectral method.
 * Adaptive_lookback is the number of lookback iterations.
 */

struct ABIP_ADAPTIVE_WORK
{
    abip_float *u_prev;
    abip_float *v_prev;
    abip_float *ut;
    abip_float *u;
    abip_float *v;
    abip_float *ut_next;
    abip_float *u_next;
    abip_float *v_next;

    abip_float *delta_ut;
    abip_float *delta_u;
    abip_float *delta_v;

    abip_int l;
    abip_int k;

    abip_float total_adapt_time;
};

```

```
static abip_int update_adapt_params
 (
  ABIPWork *w,
  abip_int iter
  )
{
    DEBUG_FUNC

    abip_float *u_prev = w->adapt->u_prev;
    abip_float *v_prev = w->adapt->v_prev;
    abip_float *ut = w->adapt->ut;
    abip_float *u = w->adapt->u;
    abip_float *v = w->adapt->v;
    abip_float *ut_next = w->adapt->ut_next;
    abip_float *u_next = w->adapt->u_next;
    abip_float *v_next = w->adapt->v_next;

    abip_float *delta_ut = w->adapt->delta_ut;
    abip_float *delta_u = w->adapt->delta_u;
    abip_float *delta_v = w->adapt->delta_v;

    abip_int l = w->adapt->l;
    abip_int n = w->n;
    abip_int m = w->m;
    abip_int k = w->adapt->k;
    abip_int i;
    abip_int j;

    abip_int status_1 = 0;
    abip_int status_2 = 0;

    abip_float tmp;
    abip_float beta_prev = 1.0;
    abip_float beta = 0.0;

    abip_float uu;
    abip_float uv;
    abip_float vv;
    abip_float utut;
    abip_float utv;
    abip_float norm_ut;
    abip_float norm_u;
    abip_float norm_v;

    abip_float alpha_SD;
    abip_float alpha_MG;
    abip_float gamma_SD;
    abip_float gamma_MG;
    abip_float alpha_cor;
    abip_float gamma_cor;
    abip_float alpha_ss;
    abip_float gamma_ss;

    memcpy(u_prev, w->u, sizeof(abip_float) * l);
    memcpy(v_prev, w->v, sizeof(abip_float) * l);

    for (i = 0; i < k; ++i)
    {
        memcpy(ut, u_prev, l * sizeof(abip_float));
        ABIP(add_scaled_array)(ut, v_prev, l, 1.0);
        ABIP(scale_array)(ut, w->stgs->rho_y, m);
        ABIP(add_scaled_array)(ut, w->h, l - 1, -ut[l - 1]);
        ABIP(add_scaled_array)(ut, w->h, l - 1, -ABIP(dot)(ut, w->g, l - 1) / (w->g_th + 1));
        ABIP(scale_array)(&(ut[m]), -1, n);
        status_1 = ABIP(solve_lin_sys)(w->A, w->stgs, w->p, ut, u_prev, iter);
        ut[l - 1] += ABIP(dot)(ut, w->h, l - 1);

        for (j = 0; j < m; ++j)
        {
            u[j] = ut[j] - v_prev[j];
        }

        for (j = m; j < l; ++j)
        {
            u[j] = w->stgs->alpha * ut[j] + (1 - w->stgs->alpha) * u_prev[j] - v_prev[j];
        }

        for(j = m; j < l; ++j)
        {
            tmp = u[j] / 2;
            u[j] = tmp + SQRTF(tmp * tmp + w->mu / beta_prev);
        }


        for (j = m; j < l; ++j)
        {
            v[j] =  v_prev[j] + (u[j] - w->stgs->alpha * ut[j] - (1 - w->stgs->alpha) * u_prev[j]);
```

```
        }


        memcpy(ut_next, u, l * sizeof(abip_float));
        ABIP(add_scaled_array)(ut_next, v, l, 1.0);
        ABIP(scale_array)(ut_next, w->stgs->rho_y, m);
        ABIP(add_scaled_array)(ut_next, w->h, l - 1, -ut_next[l - 1]);
        ABIP(add_scaled_array)(ut_next, w->h, l - 1, -ABIP(dot)(ut_next, w->g, l - 1) / (w->g_th + 1));
        ABIP(scale_array)(&(ut_next[m]), -1, n);
        status_2 = ABIP(solve_lin_sys)(w->A, w->stgs, w->p, ut_next, u, iter);
        ut_next[l - 1] += ABIP(dot)(ut_next, w->h, l - 1);

        for (j = 0; j < m; ++j)
        {
            u_next[j] = ut_next[j] - v[j];
        }

        for (j = m; j < l; ++j)
        {
            u_next[j] = w->stgs->alpha * ut_next[j] + (1 - w->stgs->alpha) * u[j] - v[j];
        }

        for(j = m; j < l; ++j)
        {
            tmp = u_next[j] / 2;
            u_next[j] = tmp + SQRTF(tmp * tmp + w->mu / beta_prev);
        }

        for (j = m; j < l; ++j)
        {
            v_next[j] =  v[j] + (u_next[j] - w->stgs->alpha * ut_next[j] - (1 - w->stgs->alpha) * u[j]);
        }

        memcpy(delta_ut, v, l * sizeof(abip_float));
        ABIP(scale_array)(delta_ut, 2.0, l);
        ABIP(add_scaled_array)(delta_ut, u_next, l, 1.0);
        ABIP(add_scaled_array)(delta_ut, u, l, -1.0);
        ABIP(add_scaled_array)(delta_ut, v_next, l, -1.0);
        ABIP(add_scaled_array)(delta_ut, v_prev, l, -1.0);

        memcpy(delta_u, u, l * sizeof(abip_float));
        ABIP(add_scaled_array)(delta_u, u_next, l, -1.0);

        memcpy(delta_v, u_next, l * sizeof(abip_float));
        ABIP(add_scaled_array)(delta_v, u, l, -1.0);
        ABIP(scale_array)(delta_v, w->stgs->alpha-1.0, l);
        ABIP(add_scaled_array)(delta_v, v_next, l, 1.0);
        ABIP(add_scaled_array)(delta_v, v, l, -1.0);

        utut = ABIP(dot)(delta_ut, delta_ut, l);
        utv = ABIP(dot)(delta_ut, delta_v, l);
        uu = ABIP(dot)(delta_u, delta_u, l);
        vv = ABIP(dot)(delta_v, delta_v, l);
        uv = ABIP(dot)(delta_u, delta_v, l);

        norm_ut = ABIP(norm)(delta_ut, l);
        norm_u = ABIP(norm)(delta_u, l);
        norm_v = ABIP(norm)(delta_v, l);

        alpha_SD = vv/utv;
        alpha_MG = utv/utut;
        gamma_SD = vv/uv;
        gamma_MG = uv/uu;
        /*
         abip_printf("alpha_SD = %3.6f, alpha_MG = %3.6f, gamma_SD = %3.6f, gamma_MG = %3.6f\n", alpha_SD, alpha_MG, gamma_SD, gamma_MG);
         */
        if (2*alpha_MG > alpha_SD)
        {
            alpha_ss = alpha_MG;
        }
        else
        {
            alpha_ss = alpha_SD - 0.5*alpha_MG;
        }

        if (2*gamma_MG > gamma_SD)
        {
            gamma_ss = gamma_MG;
        }
        else
        {
            gamma_ss = gamma_SD - 0.5*gamma_MG;
        }

```

```
        alpha_cor = utv / (norm_v*norm_ut);
        gamma_cor = uv / (norm_v*norm_u);

        if (alpha_cor > w->stgs->eps_cor && gamma_cor > w->stgs->eps_cor)
        {
            beta = SQRTF(alpha_ss*gamma_ss);
        }
        else if (alpha_cor > w->stgs->eps_cor && gamma_cor <= w->stgs->eps_cor)
        {
            beta = alpha_ss;
        }
        else if (alpha_cor <= w->stgs->eps_cor && gamma_cor > w->stgs->eps_cor)
        {
            beta = gamma_ss;
        }
        else
        {
            beta = beta_prev;
        }

        if (ABS(beta-beta_prev) > 0 && ABS(beta-beta_prev) <= w->stgs->eps_pen)
        {
            beta = (beta+beta_prev)/2;
            break;
        }
        else if (ABS(beta-beta_prev) > w->stgs->eps_pen)
        {
            beta_prev = beta;
            memcpy(u_prev, u, l * sizeof(abip_float));
            for (j = 0; j < m; ++j)
            {
                v_prev[j] = v[j];
            }
            for (j = m; j < l; ++j)
            {
                v_prev[j] = (w->mu / beta_prev) / u_prev[j];
            }
        }
        else
        {
            memcpy(u_prev, u, l * sizeof(abip_float));
            memcpy(v_prev, v, l * sizeof(abip_float));
        }
        /*
         abip_printf("beta = %3.7e, vv = %3.7e, uu = %3.7e, utv = %3.7e, uv = %3.7e, utut = %3.7e\n",
 beta, vv, uu, utv, uv, utut);
         */
    }

    w->beta = beta;

    RETURN MIN(status_1, status_2) ;
}

ABIPAdaptWork *ABIP(init_adapt)
(
 ABIPWork *w
 )
{
    DEBUG_FUNC

    ABIPAdaptWork *a = (ABIPAdaptWork *) abip_calloc(1, sizeof(ABIPAdaptWork));

    if (!a)
    {
        RETURN ABIP_NULL;
    }

    a->l = w->m + w->n + 1;

    a->k = w->stgs->adaptive_lookback;

    if (a->k <= 0)
    {
        RETURN a;
    }

    a->u_prev = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->v_prev = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->ut = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->u = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->v = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->ut_next = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->u_next = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->v_next = (abip_float *) abip_calloc(a->l, sizeof(abip_float));

    a->delta_ut = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
```

```
    a->delta_u = (abip_float *) abip_calloc(a->l, sizeof(abip_float));
    a->delta_v = (abip_float *) abip_calloc(a->l, sizeof(abip_float));

    a->total_adapt_time = 0.0;

    if (!a->u_prev || !a->v_prev || !a->ut || !a->u || !a->v || !a->ut_next || !a->u_next ||
  !a->v_next || !a->delta_ut || !a->delta_u || !a->delta_v)
    {
        ABIP(free_adapt)(a);
        a = ABIP_NULL;
    }

    RETURN a;
}

abip_int ABIP(adaptive)
(
 ABIPWork *w,
 abip_int iter
 )
{
    DEBUG_FUNC

    abip_int k = w->adapt->k;
    abip_int info;

    ABIP(timer) adapt_timer;
    if (k <= 0)
    {
        RETURN -1;
    }

    ABIP(tic)(&adapt_timer);

    info = update_adapt_params(w, iter);

    if (iter == -1)
    {
        RETURN -1;
    }

    w->adapt->total_adapt_time += ABIP(tocq)(&adapt_timer);

    RETURN info;
}

void ABIP(free_adapt)
(
 ABIPAdaptWork *a
 )
{
    DEBUG_FUNC

    if (a)
    {
        if (a->u_prev)
        {
            abip_free(a->u_prev);
        }

        if (a->v_prev)
        {
            abip_free(a->v_prev);
        }

        if (a->ut)
        {
            abip_free(a->ut);
        }

        if (a->u)
        {
            abip_free(a->u);
        }

        if (a->v)
        {
            abip_free(a->v);
        }

        if (a->ut_next)
        {
            abip_free(a->ut_next);
        }

        if (a->u_next)
        {
```

```
            abip_free(a->u_next);
        }

        if (a->v_next)
        {
            abip_free(a->v_next);
        }

        if (a->delta_ut)
        {
            abip_free(a->delta_ut);
        }

        if (a->delta_u)
        {
            abip_free(a->delta_u);
        }

        if (a->delta_v)
        {
            abip_free(a->delta_v);
        }

        abip_free(a);
    }

    RETURN;
}

char *ABIP(get_adapt_summary)
(
 const ABIPInfo *info,
 ABIPAdaptWork *a
 )
{
    DEBUG_FUNC

    char *str = (char *) abip_malloc(sizeof(char) * 64);
    sprintf(str, "\tBarzilai-Borwein spectral method: avg step time: %1.2es\n", a->total_adapt_time /
    (info->admm_iter + 1) / 1e3);

    a->total_adapt_time = 0.0;
    RETURN str;
}
```

5.98 src/cs.c File Reference

```
#include "cs.h"
#include "abip.h"
```

Functions

- cs *ABIP() cs_transpose (const cs *A, abip_int values)
- cs *ABIP() cs_compress (const cs *T)
- cs *ABIP() cs_spalloc (abip_int m, abip_int n, abip_int nnzmax, abip_int values, abip_int triplet)
- cs *ABIP() cs_spfree (cs *A)
- abip_float ABIP() cs_cumsum (abip_int *p, abip_int *c, abip_int n)
- abip_int *ABIP() cs_pinv (abip_int const *p, abip_int n)
- cs *ABIP() cs_symperm (const cs *A, const abip_int *pinv, abip_int values)

5.98.1 Function Documentation

5.98.1.1 cs_compress()

```
cs *ABIP() cs_compress (
            const cs * T )
```

Definition at line 57 of file cs.c.

5.98.1.2 cs_cumsum()

```
abip_float ABIP() cs_cumsum (
            abip_int * p,
            abip_int * c,
            abip_int n )
```

Definition at line 162 of file cs.c.

5.98.1.3 cs_pinv()

```
abip_int *ABIP() cs_pinv (
            abip_int const * p,
            abip_int n )
```

Definition at line 190 of file cs.c.

5.98.1.4 cs_spalloc()

```
cs *ABIP() cs_spalloc (
            abip_int m,
            abip_int n,
            abip_int nnzmax,
            abip_int values,
            abip_int triplet )
```

Definition at line 117 of file cs.c.

5.98.1.5 cs_spfree()

```
cs *ABIP() cs_spfree (
            cs * A )
```

Definition at line 145 of file cs.c.

5.98.1.6 cs_symperm()

```
cs *ABIP() cs_symperm (
              const cs * A,
              const abip_int * pinv,
              abip_int values )
```

Definition at line 218 of file cs.c.

5.98.1.7 cs_transpose()

```
cs *ABIP() cs_transpose (
              const cs * A,
              abip_int values )
```

Definition at line 33 of file cs.c.

5.99 cs.c

Go to the documentation of this file.

```
00001 #include "cs.h"
00002 #include "abip.h"
00003
00004 /* NB: this is a subset of the routines in the CSPARSE package by Tim Davis et. al., for the full
       package please visit
00005  * http://www.cise.ufl.edu/research/sparse/CSparse/ */
00006
00007 /* wrapper for free */
00008 static void *cs_free
00009 (
00010       void *p
00011 )
00012 {
00013       if (p)
00014       {
00015             abip_free(p);
00016       }
00017       return (ABIP_NULL);
00018 }
00019
00020 static cs *cs_done
00021 (
00022       cs *C,
00023       void *w,
00024       void *x,
00025       abip_int ok
00026 )
00027 {
00028       cs_free(w);
00029       cs_free(x);
00030       return (ok ? C : ABIP(cs_spfree)(C));
00031 }
00032
00033 cs *ABIP(cs_transpose) (const cs *A, abip_int values)
00034 {
00035     abip_int p, q, j, *Cp, *Ci, n, m, *Ap, *Ai, *w ;
00036     abip_float *Cx, *Ax ;
00037     cs *C ;
00038     m = A->m ; n = A->n ; Ap = A->p ; Ai = A->i ; Ax = A->x ;
00039     C = ABIP(cs_spalloc)(n, m, Ap [n], values && Ax, 0) ;        /* allocate result */
00040     w = abip_calloc(m, sizeof (abip_int)) ;                       /* get workspace */
00041     if (!C || !w) return (cs_done (C, w, NULL, 0)) ;        /* out of memory */
00042     Cp = C->p ; Ci = C->i ; Cx = C->x ;
00043     for (p = 0 ; p < Ap [n] ; p++) w [Ai [p]]++ ;           /* row counts */
00044     ABIP(cs_cumsum) (Cp, w, m) ;                                  /* row pointers */
00045     for (j = 0 ; j < n ; j++)
00046     {
```

```
        for (p = Ap [j] ; p < Ap [j+1] ; p++)
        {
            Ci [q = w [Ai [p]]++] = j ; /* place A(i,j) as entry C(j,i) */
            if (Cx) Cx [q] =  Ax [p];
        }
    }
    return (cs_done (C, w, NULL, 1)) ;  /* success; free w and return C */
}


cs *ABIP(cs_compress)
(
      const cs *T
)
{
      abip_int m;
      abip_int n;
      abip_int nnz;
      abip_int p;
      abip_int k;

      abip_int *Cp;
      abip_int *Ci;
      abip_int *w;
      abip_int *Ti;
      abip_int *Tj;

      abip_float *Cx;
      abip_float *Tx;

      cs *C;

      m = T->m;
      n = T->n;
      Ti = T->i;
      Tj = T->p;
      Tx = T->x;
      nnz = T->nnz;

      C = ABIP(cs_spalloc)(m, n, nnz, Tx != ABIP_NULL, 0);
      w = (abip_int *) abip_calloc(n, sizeof(abip_int));

      if (!C || !w)
      {
            return (cs_done(C, w, ABIP_NULL, 0));
      }

      Cp = C->p;
      Ci = C->i;
      Cx = C->x;

      for (k = 0; k < nnz; k++)
      {
            w[Tj[k]]++;
      }

      ABIP(cs_cumsum)(Cp, w, n);

      for (k = 0; k < nnz; k++)
      {
            Ci[p = w[Tj[k]]++] = Ti[k];
            if (Cx)
            {
                  Cx[p] = Tx[k];
            }
      }

      return (cs_done(C, w, ABIP_NULL, 1));
}

cs *ABIP(cs_spalloc)
(
      abip_int m,
      abip_int n,
      abip_int nnzmax,
      abip_int values,
      abip_int triplet
)
{
      cs *A = (cs *) abip_calloc(1, sizeof(cs));
      if (!A)
      {
              return (ABIP_NULL);
      }

      A->m = m;
      A->n = n;
```

```
    A->nnzmax = nnzmax = MAX(nnzmax, 1);
    A->nnz = triplet ? 0 : -1;

    A->p = (abip_int *) abip_malloc((triplet ? nnzmax : n + 1) * sizeof(abip_int));
    A->i = (abip_int *) abip_malloc(nnzmax * sizeof(abip_int));
    A->x = values ? (abip_float *) abip_malloc(nnzmax * sizeof(abip_float)) : ABIP_NULL;

    return ((!A->p || !A->i || (values && !A->x)) ? ABIP(cs_spfree)(A) : A);
}

cs *ABIP(cs_spfree)
(
    cs *A
)
{
    if (!A)
    {
        return (ABIP_NULL);
    }

    cs_free(A->p);
    cs_free(A->i);
    cs_free(A->x);

    return ((cs *)cs_free(A));
}

abip_float ABIP(cs_cumsum)
(
    abip_int *p,
    abip_int *c,
    abip_int n
)
{
    abip_int i;
    abip_int nnz = 0;
    abip_float nnz2 = 0;

    if (!p || !c)
    {
        return (-1);
    }

    for (i = 0; i < n; i++)
    {
        p[i] = nnz;
        nnz += c[i];
        nnz2 += c[i];
        c[i] = p[i];
    }

    p[n] = nnz;
    return (nnz2);
}

abip_int *ABIP(cs_pinv)
(
    abip_int const *p,
    abip_int n
)
{
    abip_int k;
    abip_int *pinv;

    if (!p)
    {
        return (ABIP_NULL);
    }

    pinv = (abip_int *)abip_malloc(n * sizeof(abip_int));
    if (!pinv)
    {
        return (ABIP_NULL);
    }

    for (k = 0; k < n; k++)
    {
        pinv[p[k]] = k;
    }

    return (pinv);
}

cs *ABIP(cs_symperm)
(
    const cs *A,
```

```
    const abip_int *pinv,
    abip_int values
)
{
    abip_int i;
    abip_int j;
    abip_int p;
    abip_int q;
    abip_int i2;
    abip_int j2;
    abip_int n;
    abip_int *Ap;
    abip_int *Ai;
    abip_int *Cp;
    abip_int *Ci;
    abip_int *w;

    abip_float *Cx;
    abip_float *Ax;

    cs *C;

    n = A->n;
    Ap = A->p;
    Ai = A->i;
    Ax = A->x;

    C = ABIP(cs_spalloc)(n, n, Ap[n], values && (Ax != ABIP_NULL), 0);
    w = (abip_int *) abip_calloc(n, sizeof(abip_int));

    if (!C || !w)
    {
        return (cs_done(C, w, ABIP_NULL, 0));
    }

    Cp = C->p;
    Ci = C->i;
    Cx = C->x;

    for (j = 0; j < n; j++)
    {
        j2 = pinv ? pinv[j] : j;

        for (p = Ap[j]; p < Ap[j + 1]; p++)
        {
            i = Ai[p];
            if (i > j)
            {
                continue;
            }
            i2 = pinv ? pinv[i] : i;
            w[MAX(i2, j2)]++;
        }
    }

    ABIP(cs_cumsum)(Cp, w, n);

    for (j = 0; j < n; j++)
    {
        j2 = pinv ? pinv[j] : j;

        for (p = Ap[j]; p < Ap[j + 1]; p++)
        {
            i = Ai[p];

            if (i > j)
            {
                continue;
            }

            i2 = pinv ? pinv[i] : i;
            Ci[q = w[MAX(i2, j2)]++] = MIN(i2, j2);

            if (Cx)
            {
                 Cx[q] = Ax[p];
            }
        }
    }

    return (cs_done(C, w, ABIP_NULL, 1));
}
```

5.100 src/ctrlc.c File Reference

```
#include "ctrlc.h"
```

5.101 ctrlc.c

Go to the documentation of this file.

```
/*
 * Implements signal handling (ctrl-c) for ABIP.
 *
 * Under Windows, we use SetConsoleCtrlHandler.
 * Under Unix systems, we use sigaction.
 * For Mex files, we use utSetInterruptEnabled/utIsInterruptPending.
 *
 */

#include "ctrlc.h"

#if CTRLC > 0

#ifdef MATLAB_MEX_FILE

static int istate;
void abip_start_interrupt_listener(void)
{
      istate = 0; //tSetInterruptEnabled(1);
}

void abip_end_interrupt_listener(void)
{
      utSetInterruptEnabled(istate);
}

int abip_is_interrupted(void)
{
      return 0; // utIsInterruptPending();
}

#elif(defined _WIN32 || _WIN64 || defined _WINDLL)

static int int_detected;
static BOOL WINAPI abip_handle_ctrlc(DWORD dwCtrlType)
{
      if (dwCtrlType != CTRL_C_EVENT)
      {
            return FALSE;
      }

      int_detected = 1;
      return TRUE;
}

void abip_start_interrupt_listener(void)
{
      int_detected = 0;
      SetConsoleCtrlHandler(abip_handle_ctrlc, TRUE);
}

void abip_end_interrupt_listener(void)
{
      SetConsoleCtrlHandler(abip_handle_ctrlc, FALSE);
}

int abip_is_interrupted(void)
{
      return int_detected;
}

#else /* Unix */

#include <signal.h>
static int int_detected;
struct sigaction oact;
static void abip_handle_ctrlc(int dummy)
{
      int_detected = dummy ? dummy : -1;
}
```

```
void abip_start_interrupt_listener(void)
{
      struct sigaction act;
      int_detected = 0;

      act.sa_flags = 0;
      sigemptyset(&act.sa_mask);

      act.sa_handler = abip_handle_ctrlc;
      sigaction(SIGINT, &act, &oact);
}

void abip_end_interrupt_listener(void)
{
      struct sigaction act;
      sigaction(SIGINT, &oact, &act);
}

int abip_is_interrupted(void)
{
      return int_detected;
}

#endif /* END IF MATLAB_MEX_FILE / WIN32 */

#endif /* END IF CTRLC > 0 */
```

5.102 src/linalg.c File Reference

```
#include "linalg.h"
#include <math.h>
```

Functions

- void ABIP() set_as_scaled_array (abip_float *x, const abip_float *a, const abip_float b, abip_int len)

  *compute x = b*a*
- void ABIP() set_as_sqrt (abip_float *x, const abip_float *v, abip_int len)

  compute x = sqrt(v)
- void ABIP() set_as_sq (abip_float *x, const abip_float *v, abip_int len)

  compute x = v.^2
- void ABIP() scale_array (abip_float *a, const abip_float b, abip_int len)

  *compute a *= b*
- abip_float ABIP() dot (const abip_float *x, const abip_float *y, abip_int len)

  *compute x'*y*
- abip_float ABIP() norm_sq (const abip_float *v, abip_int len)

  compute||v||_2^2
- abip_float ABIP() norm (const abip_float *v, abip_int len)

  compute ||v||_2
- abip_float ABIP() min_abs_sqrt (const abip_float *a, abip_int len, abip_float ref)

  compute square root of the minimal absolute value
- abip_float ABIP() norm_one (const abip_float *v, abip_int len)

  compute L1 norm
- abip_float ABIP() norm_one_sqrt (const abip_float *v, abip_int len)

  compute square root L1 norm
- abip_float ABIP() norm_inf (const abip_float *a, abip_int len)

  compute the infinity norm
- abip_float ABIP() norm_inf_sqrt (const abip_float *v, abip_int len)

compute square root infinity norm

- void ABIP() add_array (abip_float *a, const abip_float b, abip_int len)

    compute a .+= b

- void ABIP() add_scaled_array (abip_float *a, const abip_float *b, abip_int len, const abip_float sc)

    *compute a += sc*b*

- abip_float ABIP() norm_diff (const abip_float *a, const abip_float *b, abip_int len)

    compute ||a-b||_2^2

- abip_float ABIP() norm_inf_diff (const abip_float *a, const abip_float *b, abip_int len)

    compute max(|a-b|)

5.102.1 Function Documentation

5.102.1.1 add_array()

```
void ABIP() add_array (
            abip_float * a,
            const abip_float b,
            abip_int len )
```

compute a .+= b

Definition at line 218 of file linalg.c.

5.102.1.2 add_scaled_array()

```
void ABIP() add_scaled_array (
            abip_float * a,
            const abip_float * b,
            abip_int len,
            const abip_float sc )
```

compute a += sc*b

Definition at line 235 of file linalg.c.

5.102.1.3 dot()

```
abip_float ABIP() dot (
            const abip_float * x,
            const abip_float * y,
            abip_int len )
```

compute x'*y

Definition at line 78 of file linalg.c.

5.102.1.4 min_abs_sqrt()

```
abip_float ABIP() min_abs_sqrt (
            const abip_float * a,
            abip_int len,
            abip_float ref )
```

compute square root of the minimal absolute value

Definition at line 126 of file linalg.c.

5.102.1.5 norm()

```
abip_float ABIP() norm (
            const abip_float * v,
            abip_int len )
```

compute ||v||_2

Definition at line 115 of file linalg.c.

5.102.1.6 norm_diff()

```
abip_float ABIP() norm_diff (
            const abip_float * a,
            const abip_float * b,
            abip_int len )
```

compute ||a-b||_2^2

Definition at line 253 of file linalg.c.

5.102.1.7 norm_inf()

```
abip_float ABIP() norm_inf (
            const abip_float * a,
            abip_int len )
```

compute the infinity norm

Definition at line 182 of file linalg.c.

5.102.1.8 norm_inf_diff()

```
abip_float ABIP() norm_inf_diff (
             const abip_float * a,
             const abip_float * b,
             abip_int len )
```

compute max(|a-b|)

Definition at line 274 of file linalg.c.

5.102.1.9 norm_inf_sqrt()

```
abip_float ABIP() norm_inf_sqrt (
             const abip_float * v,
             abip_int len )
```

compute square root infinity norm

Definition at line 205 of file linalg.c.

5.102.1.10 norm_one()

```
abip_float ABIP() norm_one (
             const abip_float * v,
             abip_int len )
```

compute L1 norm

Definition at line 149 of file linalg.c.

5.102.1.11 norm_one_sqrt()

```
abip_float ABIP() norm_one_sqrt (
             const abip_float * v,
             abip_int len )
```

compute square root L1 norm

Definition at line 167 of file linalg.c.

5.102.1.12 norm_sq()

```
abip_float ABIP() norm_sq (
            const abip_float * v,
            abip_int len )
```

compute||v||_2^2

Definition at line 97 of file linalg.c.

5.102.1.13 scale_array()

```
void ABIP() scale_array (
            abip_float * a,
            const abip_float b,
            abip_int len )
```

compute a *= b

Definition at line 61 of file linalg.c.

5.102.1.14 set_as_scaled_array()

```
void ABIP() set_as_scaled_array (
            abip_float * x,
            const abip_float * a,
            const abip_float b,
            abip_int len )
```

compute x = b*a

Definition at line 9 of file linalg.c.

5.102.1.15 set_as_sq()

```
void ABIP() set_as_sq (
            abip_float * x,
            const abip_float * v,
            abip_int len )
```

compute x = v.^2

Definition at line 44 of file linalg.c.

5.102.1.16 set_as_sqrt()

```
void ABIP() set_as_sqrt (
            abip_float * x,
            const abip_float * v,
            abip_int len )
```

compute x = sqrt(v)

Definition at line 27 of file linalg.c.

5.103 linalg.c

Go to the documentation of this file.

```
#include "linalg.h"
#include <math.h>

// basic algebra operators

void ABIP(set_as_scaled_array)
(
 abip_float *x,
 const abip_float *a,
 const abip_float b,
 abip_int len
 )
{
    abip_int i;
    for (i = 0; i < len; ++i)
    {
        x[i] = b * a[i];
    }
}

void ABIP(set_as_sqrt)
(
 abip_float *x,
 const abip_float *v,
 abip_int len
 )
{
    abip_int i;
    for (i = 0; i < len; ++i)
    {
        x[i] = SQRTF(v[i]);
    }
}

void ABIP(set_as_sq)
(
 abip_float *x,
 const abip_float *v,
 abip_int len
 )
{
    abip_int i;
    for (i = 0; i < len; ++i)
    {
        x[i] = v[i]*v[i];
    }
}

void ABIP(scale_array)
(
 abip_float *a,
 const abip_float b,
 abip_int len
 )
{
    abip_int i;
    for (i = 0; i < len; ++i)
    {
        a[i] *= b;
    }
}

```

```
abip_float ABIP(dot)
(
 const abip_float *x,
 const abip_float *y,
 abip_int len
 )
{
    abip_int i;
    abip_float ip = 0.0;
    for (i = 0; i < len; ++i)
    {
        ip += x[i] * y[i];
    }
    return ip;
}

abip_float ABIP(norm_sq)
(
 const abip_float *v,
 abip_int len
 )
{
    abip_int i;
    abip_float nmsq = 0.0;
    for (i = 0; i < len; ++i)
    {
        nmsq += v[i] * v[i];
    }
    return nmsq;
}

abip_float ABIP(norm)
(
 const abip_float *v,
 abip_int len
 )
{
    return SQRTF(ABIP(norm_sq)(v, len));
}
abip_float ABIP(min_abs_sqrt)
(
 const abip_float *a,
 abip_int len,
 abip_float ref
 )
{
    abip_int i;
    abip_float tmp;
    for (i = 0; i < len; ++i)
    {
        tmp = ABS(a[i]);
        if (tmp <= ref && tmp > 0)
        {
            ref = tmp;
        }
    }
    return SQRTF(ref);
}

abip_float ABIP(norm_one)
(
 const abip_float *v,
 abip_int len
 )
{
    abip_int i;
    abip_float nmone = 0.0;
    for (i = 0; i < len; ++i)
    {
        nmone += ABS(v[i]);
    }
    return nmone;
}

abip_float ABIP(norm_one_sqrt)
(
 const abip_float *v,
 abip_int len
 )
{
    return SQRTF(ABIP(norm_one)(v, len));
}




abip_float ABIP(norm_inf)
```

```
(
 const abip_float *a,
 abip_int len
 )
{
    abip_int i;
    abip_float tmp;
    abip_float max = 0.0;
    for (i = 0; i < len; ++i)
    {
        tmp = ABS(a[i]);
        if (tmp >= max)
        {
            max = tmp;
        }
    }
    return max;
}

abip_float ABIP(norm_inf_sqrt)
(
 const abip_float *v,
 abip_int len
 )
{
    return SQRTF(ABIP(norm_inf)(v, len));
}
// ------

void ABIP(add_array)
(
 abip_float *a,
 const abip_float b,
 abip_int len
 )
{
    abip_int i;
    for (i = 0; i <len; ++i)
    {
        a[i] += b;
    }
}

void ABIP(add_scaled_array)
(
 abip_float *a,
 const abip_float *b,
 abip_int len,
 const abip_float sc
 )
{
    abip_int i;
    for (i = 0; i < len; ++i)
    {
        a[i] += sc * b[i];
    }
}

abip_float ABIP(norm_diff)
(
 const abip_float *a,
 const abip_float *b,
 abip_int len
 )
{
    abip_int i;
    abip_float tmp;
    abip_float nm_diff = 0.0;
    for (i = 0; i < len; ++i)
    {
        tmp = (a[i] - b[i]);
        nm_diff += tmp * tmp;
    }
    return SQRTF(nm_diff);
}

abip_float ABIP(norm_inf_diff)
(
 const abip_float *a,
 const abip_float *b,
 abip_int len
 )
{
    abip_int i;
    abip_float tmp;
    abip_float max = 0.0;
    for (i = 0; i < len; ++i)
```

```
    {
        tmp = ABS(a[i] - b[i]);
        if (tmp > max)
        {
            max = tmp;
        }
    }
    return max;
}
```

5.104 src/normalize.c File Reference

```
#include "abip.h"
#include "normalize.h"
#include "linalg.h"
```

Macros

- #define MIN_SCALE (1e-3)
- #define MAX_SCALE (1e3)

Functions

- void ABIP() normalize_b_c (ABIPWork ∗w)

  normalize b and c
- void ABIP() calc_scaled_resids (ABIPWork ∗w, ABIPResiduals ∗r)

  calculate the scaled residuals
- void ABIP() normalize_warm_start (ABIPWork ∗w)

  normalize the warm start solution
- void ABIP() un_normalize_sol (ABIPWork ∗w, ABIPSolution ∗sol)

  recover the optimal solution

5.104.1 Macro Definition Documentation

5.104.1.1 MAX_SCALE

```
#define MAX_SCALE (1e3)
```

Definition at line 6 of file normalize.c.

5.104.1.2 MIN_SCALE

```
#define MIN_SCALE (1e-3)
```

Definition at line 5 of file normalize.c.

5.104.2 Function Documentation

5.104.2.1 calc_scaled_resids()

```
void ABIP() calc_scaled_resids (
            ABIPWork * w,
            ABIPResiduals * r )
```

calculate the scaled residuals

Definition at line 44 of file normalize.c.

5.104.2.2 normalize_b_c()

```
void ABIP() normalize_b_c (
            ABIPWork * w )
```

normalize b and c

Definition at line 11 of file normalize.c.

5.104.2.3 normalize_warm_start()

```
void ABIP() normalize_warm_start (
            ABIPWork * w )
```

normalize the warm start solution

Definition at line 100 of file normalize.c.

5.104.2.4 un_normalize_sol()

```
void ABIP() un_normalize_sol (
            ABIPWork * w,
            ABIPSolution * sol )
```

recover the optimal solution

Definition at line 133 of file normalize.c.

5.105 normalize.c

Go to the documentation of this file.

```
#include "abip.h"
#include "normalize.h"
#include "linalg.h"

#define MIN_SCALE (1e-3)
#define MAX_SCALE (1e3)

void ABIP(normalize_b_c)
(
 ABIPWork *w
 )
{
    abip_int i;

    abip_float nm;
    abip_float *D = w->scal->D;
    abip_float *E = w->scal->E;
    abip_float *b = w->b;
    abip_float *c = w->c;

    for (i = 0; i < w->n; ++i)
    {
        c[i] /= E[i];
    }
    nm = ABIP(norm)(c, w->n);
    w->sc_c = w->scal->mean_norm_row_A / MAX(nm, MIN_SCALE);

    for (i = 0; i < w->m; ++i)
    {
        b[i] /= D[i];
    }
    nm = ABIP(norm)(b, w->m);
    w->sc_b = w->scal->mean_norm_col_A / MAX(nm, MIN_SCALE);

    ABIP(scale_array)(c, w->sc_c * w->stgs->scale, w->n);
    ABIP(scale_array)(b, w->sc_b * w->stgs->scale, w->m);
}
void ABIP(calc_scaled_resids)
(
 ABIPWork *w,
 ABIPResiduals *r
 )
{
    abip_float *D = w->scal->D;
    abip_float *E = w->scal->E;

    abip_float *u = w->u;
    abip_float *u_t = w->u_t;
    abip_float *u_prev = w->u_prev;
    abip_float tmp;

    abip_int i;
    abip_int n = w->n;
    abip_int m = w->m;

    r->res_pri = 0;
    for (i = 0; i < m; ++i)
    {
        tmp = (u[i] - u_t[i]) / (D[i] * w->sc_c);
        r->res_pri += tmp * tmp;
    }

    for (i = 0; i < n; ++i)
    {
        tmp = (u[i + m] - u_t[i + m]) / (E[i] * w->sc_b);
        r->res_pri += tmp * tmp;
    }

    tmp = u[n + m] - u_t[n + m];
    r->res_pri += tmp * tmp;
    r->res_pri = sqrt(r->res_pri);

    r->res_dual = 0;
    for (i = 0; i < m; ++i)
    {
        tmp = (u[i] - u_prev[i]) * D[i] / w->sc_c;
        r->res_dual += tmp * tmp;
    }

    for (i = 0; i < n; ++i)
    {
        tmp = (u[i + m] - u_prev[i + m]) * E[i] / w->sc_b;
```

```
        r->res_dual += tmp * tmp;
    }

    tmp = u[n + m] - u_prev[n + m];
    r->res_dual += tmp * tmp;
    r->res_dual = sqrt(r->res_dual);
}

void ABIP(normalize_warm_start)
(
 ABIPWork *w
 )
{
    abip_int i;

    abip_float *D = w->scal->D;
    abip_float *E = w->scal->E;

    abip_float *y = w->u;
    abip_float *x = &(w->u[w->m]);
    abip_float *s = &(w->v[w->m]);

    for (i = 0; i < w->n; ++i)
    {
        x[i] *= (E[i] * w->sc_b);
    }

    for (i = 0; i < w->m; ++i)
    {
        y[i] *= (D[i] * w->sc_c);
    }

    for (i = 0; i < w->n; ++i)
    {
        s[i] /= (E[i] / (w->sc_c * w->stgs->scale));
    }
}

void ABIP(un_normalize_sol)
(
 ABIPWork *w,
 ABIPSolution *sol
 )
{
    abip_int i;

    abip_float *D = w->scal->D;
    abip_float *E = w->scal->E;

    for (i = 0; i < w->n; ++i)
    {
        sol->x[i] /= (E[i] * w->sc_b);
    }

    for (i = 0; i < w->m; ++i)
    {
        sol->y[i] /= (D[i] * w->sc_c);
    }

    for (i = 0; i < w->n; ++i)
    {
        sol->s[i] *= E[i] / (w->sc_c * w->stgs->scale);
    }
}
```

5.106 src/util.c File Reference

```
#include "glbopts.h"
#include "util.h"
#include "linsys.h"
```

Functions

- void ABIP() tic (ABIP(timer) *t)

- abip_float ABIP() tocq (ABIP(timer) *t)
- abip_float ABIP() toc (ABIP(timer) *t)

    define toc function
- abip_float ABIP() str_toc (char *str, ABIP(timer) *t)

    store time consumed
- void ABIP() print_work (const ABIPWork *w)

    print the iterates
- void ABIP() print_data (const ABIPData *d)

    print some parameters
- void ABIP() print_array (const abip_float *arr, abip_int n, const char *name)

    print array
- void ABIP() free_data (ABIPData *d)

    set the memory of problem data free
- void ABIP() free_sol (ABIPSolution *sol)

    set the memory of solution free
- void ABIP() set_default_settings (ABIPData *d)

    set default setting

5.106.1 Function Documentation

5.106.1.1 free_data()

```
void ABIP() free_data (
            ABIPData * d )
```

set the memory of problem data free

Definition at line 226 of file util.c.

5.106.1.2 free_sol()

```
void ABIP() free_sol (
            ABIPSolution * sol )
```

set the memory of solution free

Definition at line 259 of file util.c.

5.106.1.3 print_array()

```
void ABIP() print_array (
             const abip_float * arr,
             abip_int n,
             const char * name )
```

print array

Definition at line 191 of file util.c.

5.106.1.4 print_data()

```
void ABIP() print_data (
             const ABIPData * d )
```

print some parameters

Definition at line 162 of file util.c.

5.106.1.5 print_work()

```
void ABIP() print_work (
             const ABIPWork * w )
```

print the iterates

Definition at line 133 of file util.c.

5.106.1.6 set_default_settings()

```
void ABIP() set_default_settings (
             ABIPData * d )
```

set default setting

Definition at line 288 of file util.c.

5.106.1.7 str_toc()

```
abip_float ABIP() str_toc (
            char * str,
            ABIP(timer) * t )
```

store time consumed

Definition at line 120 of file util.c.

5.106.1.8 tic()

```
void ABIP() tic (
            ABIP(timer) * t )
```

Definition at line 73 of file util.c.

5.106.1.9 toc()

```
abip_float ABIP() toc (
            ABIP(timer) * t )
```

define toc function

Definition at line 108 of file util.c.

5.106.1.10 tocq()

```
abip_float ABIP() tocq (
            ABIP(timer) * t )
```

Definition at line 81 of file util.c.

5.107 util.c

Go to the documentation of this file.

```
#include "glbopts.h"
#include "util.h"
#include "linsys.h"

#if (defined NOTIMER)

void ABIP(tic)
(
 ABIP(timer) *t
 ) {}

abip_float ABIP(tocq)
(
 ABIP(timer) *t
 )
{
    return NAN;
}

#elif(defined _WIN32 || _WIN64 || defined _WINDLL)

void ABIP(tic)
(
 ABIP(timer) *t
 )
{
    QueryPerformanceFrequency(&t->freq);
    QueryPerformanceCounter(&t->tic);
}

abip_float ABIP(tocq)
(
 ABIP(timer) *t
 )
{
    QueryPerformanceCounter(&t->toc);
    return (1e3 * (t->toc.QuadPart - t->tic.QuadPart) / (abip_float)t->freq.QuadPart);
}

#elif(defined __APPLE__)

void ABIP(tic)
(
 ABIP(timer) *t
 )
{
    /* read current clock cycles */
    t->tic = mach_absolute_time();
}

abip_float ABIP(tocq)
(
 ABIP(timer) *t
 )
{
    uint64_t duration;

    t->toc = mach_absolute_time();
    duration = t->toc - t->tic;

    mach_timebase_info(&(t->tinfo));
    duration *= t->tinfo.numer;
    duration /= t->tinfo.denom;

    return (abip_float)duration / 1e6;
}

#else

void ABIP(tic)
(
 ABIP(timer) *t
 )
{
    clock_gettime(CLOCK_MONOTONIC, &t->tic);
}

abip_float ABIP(tocq)
(
 ABIP(timer) *t
 )
{
```

```
    struct timespec temp;

    clock_gettime(CLOCK_MONOTONIC, &t->toc);

    if ((t->toc.tv_nsec - t->tic.tv_nsec) < 0)
    {
        temp.tv_sec = t->toc.tv_sec - t->tic.tv_sec - 1;
        temp.tv_nsec = 1e9 + t->toc.tv_nsec - t->tic.tv_nsec;
    }
    else
    {
        temp.tv_sec = t->toc.tv_sec - t->tic.tv_sec;
        temp.tv_nsec = t->toc.tv_nsec - t->tic.tv_nsec;
    }

    return (abip_float) temp.tv_sec * 1e3 + (abip_float) temp.tv_nsec / 1e6;
}

#endif
abip_float ABIP(toc)
(
 ABIP(timer) *t
 )
{
    abip_float time = ABIP(tocq)(t);
    abip_printf("time: %8.4f milli-seconds.\n", time);
    return time;
}
abip_float ABIP(str_toc)
(
 char *str,
 ABIP(timer) *t
 )
{
    abip_float time = ABIP(tocq)(t);
    abip_printf("%s - time: %8.4f milli-seconds.\n", str, time);
    return time;
}
void ABIP(print_work)
(
 const ABIPWork *w
 )
{
    abip_int i;
    abip_int l = w->n + w->m;

    abip_printf("\n u_t is \n");
    for (i = 0; i < l; i++)
    {
        abip_printf("%f\n", w->u_t[i]);
    }

    abip_printf("\n u is \n");
    for (i = 0; i < l; i++)
    {
        abip_printf("%f\n", w->u[i]);
    }

    abip_printf("\n v is \n");
    for (i = 0; i < l; i++)
    {
        abip_printf("%f\n", w->v[i]);
    }
}
void ABIP(print_data)
(
 const ABIPData *d
 )
{
    abip_printf("m = %i\n", (int)d->m);
    abip_printf("n = %i\n", (int)d->n);

    abip_printf("max_ipm_iters = %i\n", (int)d->stgs->max_ipm_iters);
    abip_printf("max_admm_iters = %i\n", (int)d->stgs->max_admm_iters);

    abip_printf("verbose = %i\n", (int)d->stgs->verbose);
    abip_printf("normalize = %i\n", (int)d->stgs->normalize);
    abip_printf("warm_start = %i\n", (int)d->stgs->warm_start);
    abip_printf("adaptive = %i\n", (int)d->stgs->adaptive);
    abip_printf("adaptive_lookback = %i\n", (int)d->stgs->adaptive_lookback);

    abip_printf("eps = %4f\n", d->stgs->eps);
    abip_printf("alpha = %4f\n", d->stgs->alpha);
    abip_printf("rho_y = %4f\n", d->stgs->rho_y);
    abip_printf("scale = %4f\n", d->stgs->scale);

    abip_printf("eps_cor = %4f\n", d->stgs->eps_cor);
```

```
    abip_printf("eps_pen = %4f\n", d->stgs->eps_pen);

}
void ABIP(print_array)
(
 const abip_float *arr,
 abip_int n,
 const char *name
 )
{
    abip_int i;
    abip_int j;
    abip_int k = 0;

    abip_int num_on_one_line = 10;

    abip_printf("\n");
    for (i = 0; i < n / num_on_one_line; ++i)
    {
        for (j = 0; j < num_on_one_line; ++j)
        {
            abip_printf("%s[%li] = %4f, ", name, (long)k, arr[k]);
            k++;
        }
        abip_printf("\n");
    }

    for (j = k; j < n; ++j)
    {
        abip_printf("%s[%li] = %4f, ", name, (long)j, arr[j]);
    }

    abip_printf("\n");
}

void ABIP(free_data)
(
 ABIPData *d
 )
{
    if (d)
    {
        if (d->b)
        {
            abip_free(d->b);
        }

        if (d->c)
        {
            abip_free(d->c);
        }

        if (d->stgs)
        {
            abip_free(d->stgs);
        }

        if (d->A)
        {
            ABIP(free_A_matrix)(d->A);
        }

        abip_free(d);
    }
}
void ABIP(free_sol)
(
 ABIPSolution *sol
 )
{
    if (sol)
    {
        if (sol->x)
        {
            abip_free(sol->x);
        }

        if (sol->y)
        {
            abip_free(sol->y);
        }

        if (sol->s)
        {
            abip_free(sol->s);
        }

```

```
        abip_free(sol);
    }
}

void ABIP(set_default_settings)
(
 ABIPData *d
 )
{
    d->stgs->max_ipm_iters = MAX_IPM_ITERS;
    d->stgs->max_admm_iters = MAX_ADMM_ITERS;
    d->stgs->eps = EPS;
    d->stgs->alpha = ALPHA;
    d->stgs->cg_rate = CG_RATE;

    d->stgs->normalize = NORMALIZE;
    d->stgs->scale = SCALE;
    d->stgs->rho_y = RHO_Y;
    d->stgs->sparsity_ratio = SPARSITY_RATIO;

    d->stgs->adaptive = ADAPTIVE;
    d->stgs->eps_cor = EPS_COR;
    d->stgs->eps_pen = EPS_PEN;
    d->stgs->adaptive_lookback = ADAPTIVE_LOOKBACK;

    d->stgs->dynamic_x = 0.8;
    d->stgs->dynamic_eta = 1.1;

    d->stgs->restart_fre = 1000;
    d->stgs->restart_thresh = 100000;

    d->stgs->origin_rescale = 0;
    d->stgs->pc_ruiz_rescale = 1;
    d->stgs->qp_rescale = 0;
    d->stgs->ruiz_iter = 10;
    d->stgs->hybrid_mu = 1; // whether use hybrid mu strategy
    d->stgs->dynamic_sigma = -1.0;
    d->stgs->hybrid_thresh = 1000; // the threshold to switch mu strategy
    d->stgs->dynamic_sigma_second = 0.5;

    d->stgs->half_update = 0; // whether use half update
    d->stgs->avg_criterion = 0;

    d->stgs->verbose = VERBOSE;
    d->stgs->warm_start = WARM_START;
}
```

Index