

### Category 1: invalid operation exception

1. The function `gsl_sf_conicalP_xlt1_large_neg_mu_e` throws an invalid exception when its input  $\tau=1.0$ ,  $\mu=2.0$ ,  $x=3.0$  at line 221 in file `/gsl-2.7.1/specfunc/legendre_con.c`:  $p = x/\sqrt{\beta_2^2(1.0-x^2) + 1.0}$ ;

Because the parameter of `sqrt` cannot be negative numbers, invalid exception occurs in this occasion.

```
214 int
215 gsl_sf_conicalP_xlt1_large_neg_mu_e(double mu, double tau, double x,
216                                         gsl_sf_result * result, double * ln_multiplier)
217 {
218     double beta = tau/mu;
219     double beta2 = beta*beta;
220     double S    = beta * acos((1.0-beta2)/(1.0+beta2));
221     double p    = x/sqrt(beta2*(1.0-x*x) + 1.0);
    1 Argument value is out of valid domain [0, inf) for function call to sqrt
222     gsl_sf_result lg_mup1;
223     int lg_stat = gsl_sf_lngamma_e(mu+1.0, &lg_mup1);
224     double ln_pre_1 = 0.5*mu*(S - log(1.0+beta2) + log((1.0-p)/(1.0+p))) - lg_mup1.val;
225     double ln_pre_2 = -0.25 * log(1.0 + beta2*(1.0-x));
226     double ln_pre_3 = -tau * atan(p*beta);
227     double ln_pre = ln_pre_1 + ln_pre_2 + ln_pre_3;
228     double sum   = 1.0 - olver_U1(beta2, p)/mu + olver_U2(beta2, p)/(mu*mu);
229
230     if(sum == 0.0) {
231         result->val = 0.0;
232         result->err = 0.0;
233         *ln_multiplier = 0.0;
234         return GSL_SUCCESS;
235     }
236     else {
237         int stat_e = gsl_sf_exp_mult_e(ln_pre, sum, result);
238         if(stat_e != GSL_SUCCESS) {
239             result->val = sum;
240             result->err = 2.0 * GSL_DBL_EPSILON * fabs(sum);
241             *ln_multiplier = ln_pre;
242         }
243         else {
244             *ln_multiplier = 0.0;
245         }
246     }
247 }
248 }
```

2. The function `gsl_cdf_laplace_Qinv` throws an invalid exception when its input  $Q=-0.5$  at line 67 in file `/gsl-2.7.1/cdf/laplaceinv.c`:  $x = -a * \log(2*Q)$ ;

Because the parameter of `log` cannot be negative numbers, invalid exception occurs in this occasion.

The screenshot shows a static code analysis interface. On the left, a tree view lists several floating-point math errors, including "Argument value is out of valid domain [0, inf)". A red arrow points from this error to a specific line of code in the center. The code is part of the `gsl_cdf_laplace_Pinv` function:

```

23 #include <gsl/gsl_cdf.h>
24
25 double
26 gsl_cdf_laplace_Pinv (const double P, const double a)
27 {
28     double x;
29
30     if (P == 1.0)
31     {
32         return GSL_POSINF;
33     }
34     else if (P == 0.0)
35     {
36         return GSL_NEGINF;
37     }
38     if (P < 0.5)
39     {
40         x = a * log(2*P);
41     }
42     else
43     {
44         x = -a * log(2*(1-P));
45     }
46     return x;
47 }
48
49 double
50 gsl_cdf_laplace_Qinv (const double Q, const double a)
51 {
52     double x;
53
54     if (Q == 0.0)
55     {
56         return GSL_POSINF;
57     }
58     else if (Q == 1.0)
59     {
60         return GSL_NEGINF;
61     }
62     if (Q < 0.5)
63     {
64         x = -a * log(2*Q);
65     }
66     else
67     {
68         x = a * log(2*(1-Q));
69     }
70     return x;
71 }

```

A red box highlights the line `x = -a * log(2*Q);`. A tooltip below it says: "Argument value is out of valid domain [0, inf) for function call to log".

### Category 2: overflow

4. The function `gsl_sf_conicalP_xlt1_large_neg_mu_e` throws a overflow exception when its input  $\tau = 1.23e189$ ,  $\mu=1e-2$  at line 219 in file `/gsl-2.7.1/specfunc/legendre_con.c`: `double beta2 = beta*beta;`

The screenshot shows a static code analysis interface. A red arrow points from a warning about overflow risk to a specific line of code in the `gsl_sf_conicalP_xlt1_large_neg_mu_e` function:

```

185
186     if (0)
187     static double elver_U3(double beta2, double p)
188     {
189         double beta = beta2*beta2;
190         double beta1 = beta*beta*beta2;
191         double opbeta2 = (1.0+beta2)/(1.0+beta2);
192         double den = 39813120.0 * gpB2*gpB2;
193         double poly1 = gsl_poly_eval(U3c2, 6, p);
194         double poly2 = gsl_poly_eval(U3c2, 6, p);
195         double poly3 = gsl_poly_eval(U3c3, 8, p);
196         double poly4 = gsl_poly_eval(U3c3, 10, p);
197         double poly5 = gsl_poly_eval(U3c5, 12, p);
198         double poly6 = gsl_poly_eval(U3c6, 12, p);
199
200         return (p-1.0)*(
201             1215.0*poly1 + 324.0*beta2*poly2
202             + 54.0*beta4*poly3 + 12.0*beta4*poly4
203             ) / den;
204     }
205     endif /* 0 */
206
207     /* large negative mu asymptotic
208     * p[-1/2 + i*tau], mu > Inf
209     * |x| < 1
210     * [Dunster, Proc. Roy. Soc. Edinburgh 119A, 311 (1991), p. 326]
211     */
212     int
213     elver_U3(double mu, double tau, double x,
214             gsl_sf_result * result, double * in_multiplier)
215     {
216         double beta = tau*mu;
217         double beta2 = beta*beta;
218         double S = beta2 * acos((1.0-beta2)/(1.0*beta2));
219         double lg_mupl = arg(beta2*(1.0-x));
220         gsl_sf_result lg_mupl;
221         int lg_stat = gsl_sf_lgamma(mu, 0, &lg_mupl);
222         double ln_prel = 0.5*mu*(log(1.0-beta2) + log((1.0-p)/(1.0*p)) - lg_mupl.val);
223         double ln_prel2 = 0.5*mu*(log(1.0+beta2*(1.0-x)));
224         double ln_prel3 = -tau * stanh(p*beta);
225         double ln_prel4 = -tau * stanh(p*beta);
226         double ln_prel5 = ln_prel1 + ln_prel2 + ln_prel3;
227         double sum = 1.0 - elver_U1(beta2, p)/mu + elver_U2(beta2, p)/(mu*mu);
228     }
229     #endif /* rhankronalha alpha MulDi */

```

A red box highlights the line `double S = beta2 * acos((1.0-beta2)/(1.0*beta2));`. A tooltip below it says: "large negative mu asymptotic".

### Category 3: underflow

5. The function `gsl_sf_bessel_Jnu_asympx_e` throws an underflow exception when its input  $\nu=1.23e-189$ ,  $x=1.0$  at line 217 in file `/gsl-2.7.1/specfunc/bessel.c`: `double mu = 4.0*nu*nu;`

未被判定  未指定责任人  显示路径箭头

展示缺陷代码  展示缺陷介绍  展示评论信息(0)

```

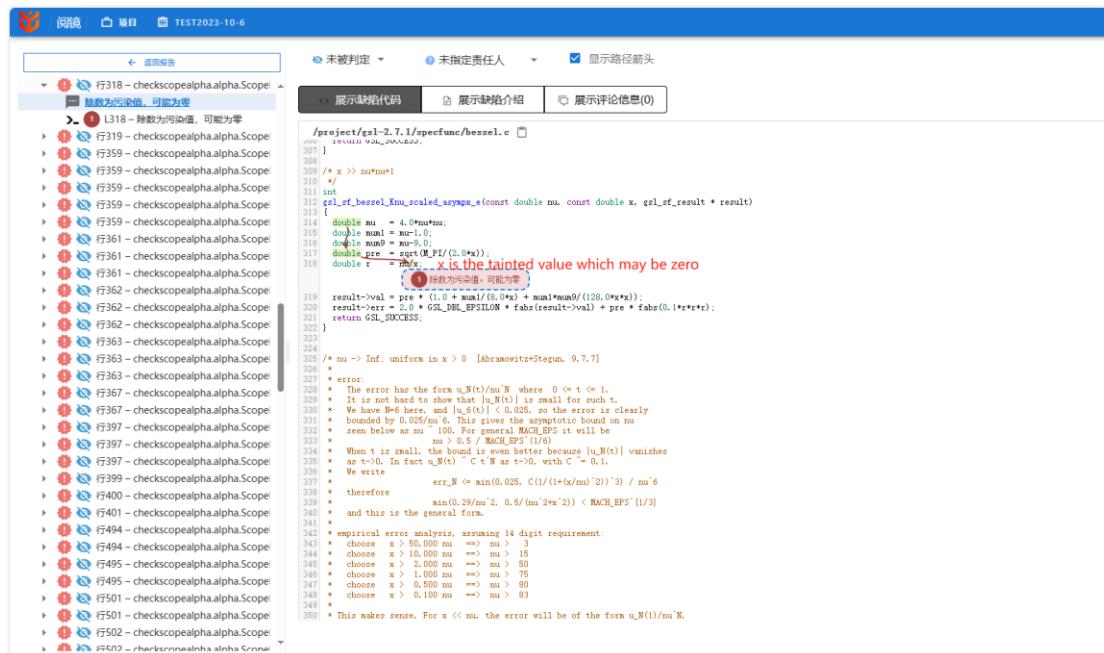
/project/gsl-2.7.1/specfunc/legendre_con.c
304 */
305 int
306 gsl_sf_conicalP_xgt1_neg_mu_largetau_e(const double mu, const double tau,
307                                         const double x, double acosh_x,
308                                         gsl_sf_result * result, double * ln_multiplier)
309 {
310     double xi = acosh_x;
311     double ln_xi_pre;
312     double ln_p_re;
313     double sumA, sumB, sum;
314     double arg;
315     gsl_sf_result J_mu1;
316     gsl_sf_result J_mu;
317     double J_mu1;
318
319     if(xi < GSL_ROOT4_DBL_EPSILON) {
320         ln_xi_pre = -xi*xi/6.0; /* log(1.0 - xi*xi/6.0) */
321     }
322     else {
323         gsl_sf_result lnshxi;
324         gsl_sf_lnsinh_e(xi, &lnshxi);
325         ln_xi_pre = log(xi) - lnshxi.val; /* log(xi/sinh(xi)) */
326     }
327
328     ln_p_re = 0.5*ln_xi_pre - mu*log(tau);
329
330     arg = tau*xi;
331
332     gsl_sf_bessel_Jnu_e(mu + 1.0, arg, &J_mu1);
333     /* Calling 'gsl_sf_bessel_Jnu_e' >
334     gsl_sf_bessel_Jnu_e(mu, arg, &J_mu);
335     J_mu1 = -J_mu1.val + 2.0*mu/arg*J_mu.val; /* careful of mu < 1 */
336
337     sumA = 1.0 - olver_A1_xi(-mu, xi)/(tau*tau);
338     sumB = olver_B0_xi(-mu, xi);
339     sum = J_mu.val * sumA - xi/tau * J_mu1 * sumB;
340
341     if(sum == 0.0) {
342         result->val = 0.0;
343         result->err = 0.0;
344         *ln_multiplier = 0.0;
345         return GSL_SUCCESS;
346     }
347     else {
348         int stat_e = gsl_sf_exp_mult_e(ln_p_re, sum, result);
349         if(stat_e != GSL_SUCCESS) {
350             result->val = sum;
351             result->err = 2.0 * GSL_DBL_EPSILON * fabs(sum);
352         }
353         else {
354             *ln_multiplier = ln_p_re;
355         }
356     }
357 }
```

```
project/gsl-2.7.1/specfunc/bessel_Jnu.c
```

155 int n;  
156 for(n=N; n>0; n--) {  
157 Jnm1 = 2.0\*(mu+n)\* x / Jn - Jnp1;  
158 Jnp1 = Jn;  
159 Jn = Jnm1;  
160 }  
161 Jmup1\_Jmu = Jnp1/Jn;  
162 sgn\_Jmu = GSL\_SIGN(Jn);  
163 Jmuprime\_Jmu = mu/x - Jmup1\_Jmu;  
164  
165 gamma = (P - Jmuprime\_Jmu)/Q;  
166 Jmu = sgn\_Jmu \* sqrt(2.0/(M\_PI\*x)) / (Q + gamma\*(P - Jmuprime\_Jmu));  
167  
168 result->val = Jmu \* (sgn\_Jmu \* GSL\_SQRT\_DBL\_MIN) / Jn;  
169 result->err = 2.0 \* GSL\_DBL\_EPSILON \* (N + 2.0) \* fabs(result->val);  
170  
171 return GSL\_ERROR\_SELECT\_2(stat\_CF2, stat\_CF1);  
172 }  
173 }  
174 }  
175 }  
176 int  
2 < Entered call from 'gsl\_sf\_conicalP\_xt1\_neg\_mu\_largetau\_e' >  
177 gsl\_sf\_Nbessel\_Jnu\_e(const double nu, const double x, gsl\_sf\_result \* result)  
178 {  
/\* CHECK\_POINTER(result) \*/  
180  
181 if(x <= 0.0) {  
182 DOMAIN\_ERROR(result);  
183 }  
184 else if (nu < 0.0) {  
185 int Jstatus = gsl\_sf\_bessel\_Jnupos\_e(-nu, x, result);  
186 double Jval = result->val;  
187 double Jerr = result->err;  
188 int Ystatus = gsl\_sf\_bessel\_Ynupos\_e(-nu, x, result);  
189 double Yval = result->val;  
190 double Yerr = result->err;  
191 /\* double s = sin(M\_PI\*nu), c = cos(M\_PI\*nu); \*/  
192 int sinstatus = gsl\_sf\_sin\_pi\_e(nu, result);  
193 double s = result->val;  
194 double c = result->val;  
195 int cosstatus = gsl\_sf\_cos\_pi\_e(nu, result);  
196 double c = result->val;  
197 double cerr = result->err;  
198 result->val = s\*Yval + c\*Jval;  
199 result->err = fabs(c\*Yerr) + fabs(s\*cJerr) + fabs(cerr\*Yval) + fabs(serr\*Jval);  
200 return GSL\_ERROR\_SELECT\_4(Jstatus, Ystatus, sinstatus, cosstatus);  
201 }  
202 else return gsl\_sf\_bessel\_Jnupos\_e(nu, x, result);  
3 < Calling 'gsl\_sf\_bessel\_Jnupos\_e' >

#### Category 4: division-by-zero

6. The function `gsl_sf_bessel_Knu_scaled_asympx_e` throws an division-by-zero floating-point exception when its input  $x=0.0$  at line 318 in file `/gsl-2.7.1/specfunc/bessel.c`: double  $r = \nu/x;$



#### Category 5: type conversion

7. The function `gsl_ran_gamma_knuth` throws an overflow caused by type conversion exception when its input  $a=1.23e189$  at line 44 in file `/gsl-2.7.1/randist/gamma.c`: `unsigned int na = floor (a);`

阅读 项目 11-10

安全等级:未指定  
行44 - checkscopealpha.alpha.FloatToInt  
初始化情况下转换  
L44 - 初始化情况下转换

未被判定 未指定责任人 显示路径箭头

展示缺陷代码 展示缺陷介绍 展示评论信息(0)

/project/gsl-2.7.1/randist/gamma.c

```
40 double
41 gsl_ran_gamma_knuth (const gsl_rng * r, const double a, const double b)
42 {
43 /* assume a > 0 */
44 unsigned int na = floor (a);
45 // 初始化情况下转换
46 if(a >= UINT_MAX)
47 {
48     return b * (gamma_large (r, floor (a)) + gamma_frac (r, a - floor (a)));
49 }
50 else if (a == na)
51 {
52     return b * gsl_ran_gamma_int (r, na);
53 }
54 else if (na == 0)
55 {
56     return b * gamma_frac (r, a);
57 }
58 else
59 {
60     return b * (gsl_ran_gamma_int (r, na) + gamma_frac (r, a - na));
61 }
62 }
63
64 double
65 gsl_ran_gamma_int (const gsl_rng * r, const unsigned int a)
66 {
67     if (a < 12)
68     {
69         unsigned int i;
70         double prod = 1;
71
72         for (i = 0; i < a; i++)
73         {
74             prod *= gsl_rng_uniform_pos (r);
75         }
76
77         /* Note: for 12 iterations we are safe against underflow, since
78            the smallest positive random number is 0(2^-32). This means
79            the smallest possible product is 2^(-12*32) = 10^-116 which
80            is within the range of double precision. */
81
82     }
83
84     return -log (prod);
85 }
```