Life insurance calculation

Effiziente Programme (WS2011)

Stefan Neubauer Josef Eisl Bernhard Urban

http://www.complang.tuwien.ac.at/anton/lvas/effizienz-abgaben/2011w/bjs

January 20, 2012

Input- and Output Data

- Input

  Gender/Age 0,1; 0 . . . 80
  Additional risk 0, 0 . . . 0.3 (`double`), few > 0.3
  Sum insured 90000 . . . 190000 (`double`)
  Duration (`n`) 10 . . . 50 (`int`)
  Interest rate 0.025, 0.035, few others
- Output
  - Yearly premium rate
  - Reserve values $(1 \ldots n)$

$$GP = \frac{{}_nA_x + {}_nE_x + \gamma\ddot{a}_{x,n}}{\ddot{a}_{x,n} - \alpha} \quad V_t = {}_{n-t}A_{x+t} + {}_{n-t}E_{x+t} - GP \cdot \ddot{a}_{x+t,n-t}$$

$\Rightarrow_{3(n+1)}$

$${}_nE_x = {}_np_x \cdot v^n \quad {}_nA_x = \sum_{i=0}^{n-1} {}_ip_x \cdot q_{x+i} \cdot v^{i+1} \quad \ddot{a}_{x,n} = \sum_{i=0}^{n-1} {}_ip_x \cdot v^i$$

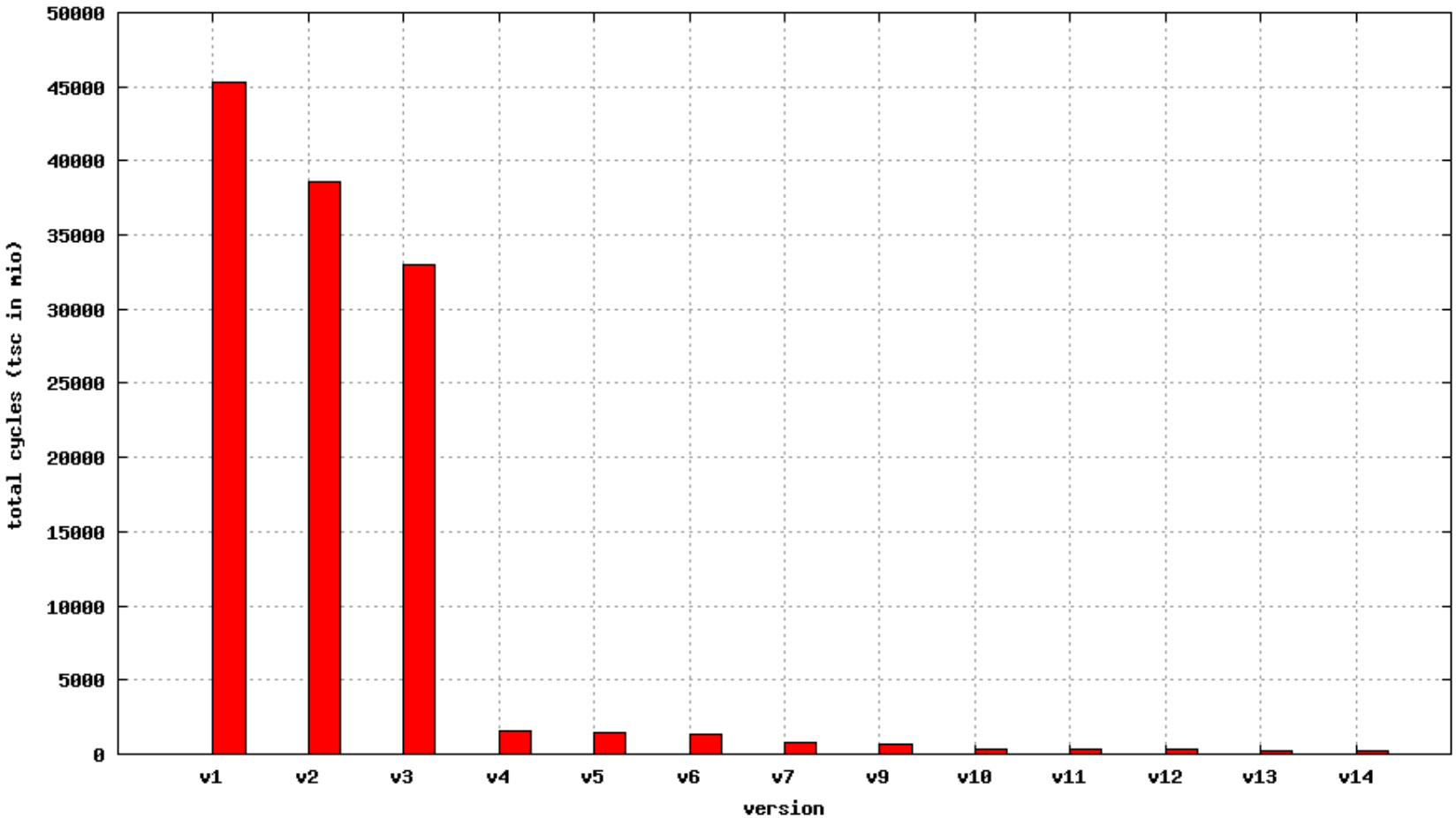
$\Rightarrow_{3n}$

$${}_np_x = \prod_{i=0}^{n-1} p_{x+i}$$

$\Rightarrow_n$ $O(n^3)$

$$p_x = 1 - q_x$$

$$q_x = min(1, Mort_{gender,x} + addrisk)$$


50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
total cycles (tsc in mio)
v1
v2
v3
v4
v5
v6
v7
v9
v10
v11
v12
v13
v14
version

v1: basic version

```
static double nAx(int x,int gender,double irate,double addrisk,int n) {
    int i; double p; double ret = 0;
    for (i = 0; i < n; i++) {
        p = npx(x, gender, addrisk, i);
        ret += p * qx(x+i, gender, addrisk) * vn(irate, i+1);
    }
    return ret;
}
static double npx(int x, int gender, double addrisk, int n) {
    int t;
    double p=1;
    for (t=0; t<n; t++) {
        p *= px(x+t, gender, addrisk);
    }
    return p;
}
static double qx(int x, int gender, double addrisk) {
    if (x>100) return 1;
    double q = mortality[gender][x] + addrisk;
    return q > 1 ? 1 : q;
}
static double vn(double irate, int n) {
    return 1 / pow(1+irate, n);
}
```

execution time	17040ms
tsc	45313372620
branch mispred.	167343838
L2 accesses	697282
instructions	66651347197
tsc (-O0 vs. -O3)	-57.97 %

v2: loop fusion

```
static double nAx(int x, ...) { ... }
static double nEx(int x, ...) { ... }
static double aexn(int x, ...) { ... }
```

v2: loop fusion

```
static double nAx(int x, ...) { ... }
static double nEx(int x, ...) { ... }
static double aexn(int x, ...) { ... }
```

```
static void barwert(..., double *ebw, double *ebw, double *rbw) {
    ...
    /* nAx, aexn */
    for (i=0; i<n; i++) {
        q = qx(x+i, gender, addrisk);
        p = npx(x, gender, addrisk, i);
        *abw += p * q * vn(irate, i+1);
        *rbw += p * vn(irate, i);
    }
    /* nEx */
    *ebw = npx(x, gender, addrisk, n) * vn(irate, n);
}
```

v2: loop fusion — statistics

execution time	14507ms	-14.87 %
tsc	38580847528	-14.86 %
branch mispred.	95995014	-42.64 %
L2 accesses	154172	-77.89 %
instructions	54207328697	-18.67 %

v3: incremental computation

```
   /* nAx, aexn */
for (i=0; i<n; i++) {
     q = qx(x+i,gender,addrisk);
     p = npx(x,gender,addrisk,i);
     *abw += p*q*vn(irate,i+1);
     *rbw += p * vn(irate, i);
}
/* nEx */
*ebw = npx(x,gender, addrisk, n)
       * vn(irate, n);
```

v3: incremental computation

```
   /* nAx, aexn */
for (i=0; i<n; i++) {
     q = qx(x+i,gender,addrisk);
     p = npx(x,gender,addrisk,i);
     *abw += p*q*vn(irate,i+1);
     *rbw += p * vn(irate, i);
}
/* nEx */
*ebw = npx(x,gender, addrisk, n)
       * vn(irate, n);
```

```
   /* nAx, aexn */
p = 1;
for (i=0; i<n; i++) {
     q = qx(x+i,gender,addrisk);
     *abw += p*q*vn(irate,i+1);
     *rbw += p*vn(irate,i);
     p *= (1 - q);
}
/* nEx */
*ebw = p * vn(irate, n);
```

v3: incremental computation

```
   /* nAx, aexn */
for (i=0; i<n; i++) {
     q = qx(x+i,gender,addrisk);
     p = npx(x,gender,addrisk,i);
     *abw += p*q*vn(irate,i+1);
     *rbw += p * vn(irate, i);
}
/* nEx */
*ebw = npx(x,gender, addrisk, n)
       * vn(irate, n);
```

```
   /* nAx, aexn */
p = 1;
for (i=0; i<n; i++) {
     q = qx(x+i,gender,addrisk);
     *abw += p*q*vn(irate,i+1);
     *rbw += p*vn(irate,i);
     p *= (1 - q);
}
/* nEx */
*ebw = p * vn(irate, n);
```

execution time	12422ms	-14.37 %
tsc	33039019552	-14.36 %
branch mispred.	40227572	-58.09 %
L2 accesses	130840	-15.13 %
instructions	40742577380	-24.84 %

v4: precompute vn(...)

```
static double vn(double irate, int n) {
    return 1 / pow(1 + irate, n);
}
```

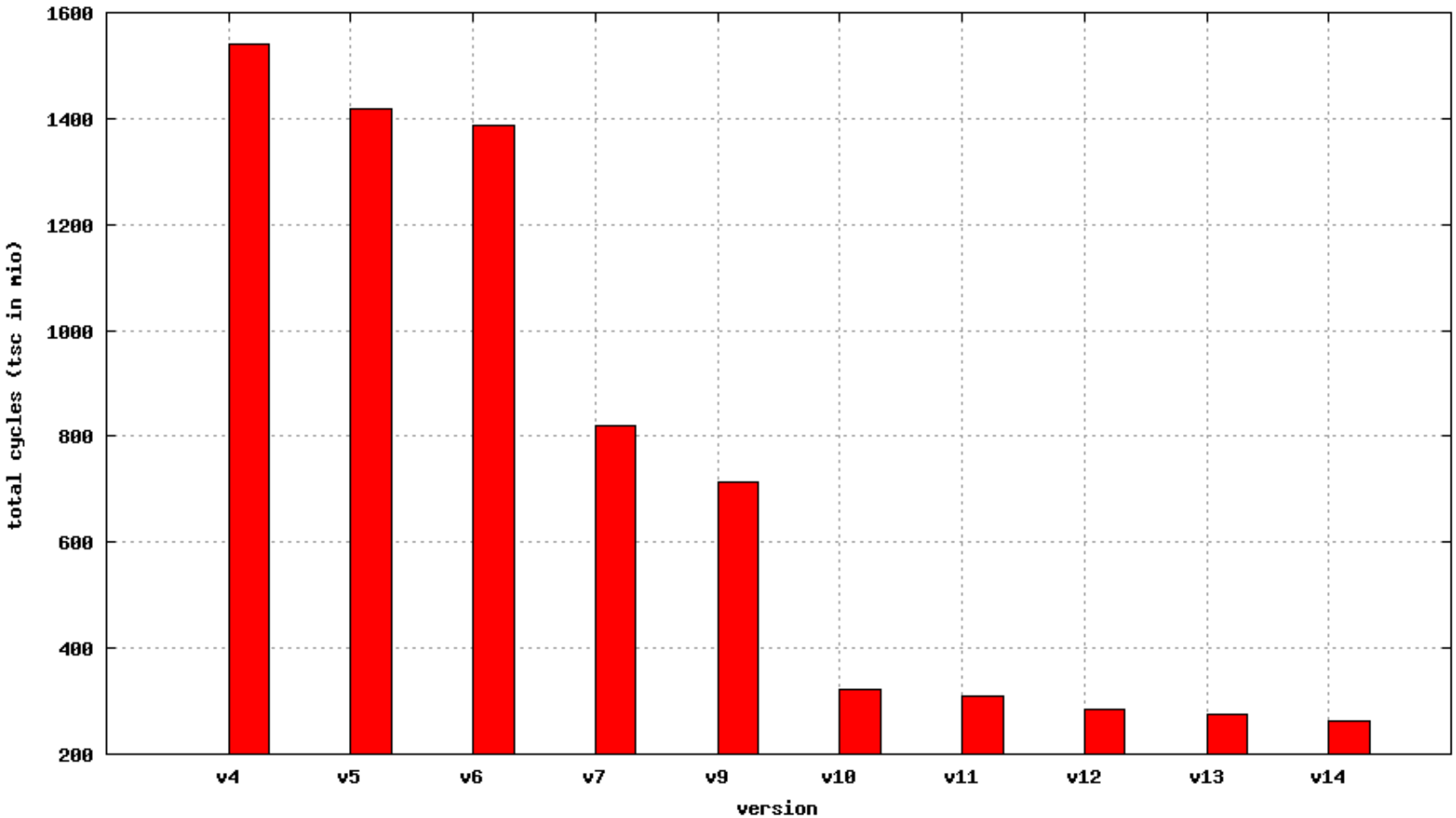
v4: precompute vn(...)

```
static double vn(double irate, int n) {
    return 1 / pow(1 + irate, n);
}
```

```
static double vn(int n) {
   return vn_precomputed[n];
}
static void precompute_vn_irate(double irate, int n) {
    int i;
    for (i=0; i<=n; i++) {
        vn_precomputed[i] = 1 / pow(1 + irate, i);
    }
}
...
```

v4: precompute `vn(...)` statistics

execution time	576ms	-95.36 %
tsc	1539598256	-95.34 %
branch mispred.	4099343	-89.81 %
L2 accesses	32512	-75.15 %
instructions	2593402053	-93.63 %


1600
1400
1200
1000
800
600
400
200
total cycles (tsc in mio)
v4
v5
v6
v7
v9
v10
v11
v12
v13
v14
version

v5: move conditional out of loop

```
static double qx(int x, int gender, double addrisk) {
    if (x>100) {
        return 1;
    }
    double q = mortality[gender][x] + addrisk;
    return q > 1 ? 1 : q;
}
...
```

v5: move conditional out of loop

```
static double qx(int x, int gender, double addrisk) {
    if (x>100) {
        return 1;
    }
    double q = mortality[gender][x] + addrisk;
    return q > 1 ? 1 : q;
}
...
```

```
static double qx(int x, int gender, double addrisk) {
    double q = mortality[gender][x] + addrisk;
    return q > 1 ? 1 : q;
}
...
```

v5: move conditional out of loop statistics

execution time	528ms	-8.33 %
tsc	1418438159	-7.87 %
branch mispred.	3791906	-7.50 %
L2 accesses	31988	-1.61 %
instructions	2233707387	-13.87 %

v6: specialization for addrisk=0

```
static double qx(int x, int gender, double addrisk) {
    double q = mortality[gender][x] + addrisk;
    return q > 1 ? 1 : q;
}
static void barwert(...) { ... }
```

v6: specialization for addrisk=0

```
static double qx(int x, int gender, double addrisk) {
    double q = mortality[gender][x] + addrisk;
    return q > 1 ? 1 : q;
}
static void barwert(...) { ... }
```

```
static double qx_addrisk0(int x, int gender) {
    return mortality[gender][x];
}
static void barwert_addrisk0(...) { ... }
```

v6: specialization for `addrisk=0` statistics

execution time	519ms	-1.70 %
tsc	1386875354	-2.23 %
branch mispred.	3610444	-4.79 %
L2 accesses	34353	+7.39 %
instructions	2156008835	-3.48 %

v7: precompute *barwert* for addrisk=0

```
typedef struct {
    double irate;
    /* indizes: gender (0/1), age (0..130), duration (0..50) */
    struct_barwert_result results[2][131][51];
} struct_barwert_addrisk0_cache;
static struct_barwert_addrisk0_cache barwert_caches[2];

precompute_barwert_addrisk0() { /* ... */ }
static void barwert_addrisk0_fromCache(...) {
    /* ... */
    pcache = &barwert_caches[ind_cache];
    if (pcache->irate==irate) {
        presult = &(pcache->results[gender][x][n]);
        *abw = presult->abw;
        *ebw = presult->ebw;
        *rbw = presult->rbw;
        return;
    }
    /* ... */ barwert_addrisk0(...) /* ... */
}
```

v7: precompute *barwert* for `addrisk=0` statistics

execution time	300ms	-42.20 %
tsc	820870385	-40.81 %
branch mispred.	2067261	-42.74 %
L2 accesses	2784181	+8004.62 %
instructions	1276234883	-40.81 %

v9: precompute, compile-time initialization

```
static double irate_cache[2] = { 0.025, 0.035 };
/* indizes: irate-ind, gender (0/1), age (0..80),   */
/* results: duration 10..50, duration+2 values per duration */
double results_cache[2][2][81][1312] =
#include "pre.txt" /* ~8 MB */
;
static double doit_addrisk0_withcache (...) {
    /* ... */
    double *pre = results_cache[ind_cache][gender][age];
    start = (n+1)*(n+2)/2 - 66;
    return doit_addrisk0_postcache(pre+start, n, vsum, res_out);
}

pre.txt:
{{{{
     /* c0 m x0 */
     /* c0 m x0 n10*/    prem, res0, ..., res10
     /* c0 m x0 n11*/ , prem, res0, ..., res10, res11
     ...
}}}}
/* conversion specification %A very useful */
```

v9: precompute, compile-time initialization statistics

execution time	261ms	-13.00 %
tsc	714441455	-12.97 %
branch mispred.	1807373	-12.57 %
L2 accesses	434750	-84.38 %
instructions	1070242984	-16.14 %

v10: more caching: `vn(...)`

```
/* indizes: irate-ind, x (0..50) */
static double vn_cache[2][51] =
#include "prev.txt"
;
```

v10: more caching: `vn(...)`

```
/* indizes: irate-ind, x (0..50) */
static double vn_cache[2][51] =
#include "prev.txt"
;
```

execution time	119ms	-54.41 %
tsc	322681488	-54.83 %
branch mispred.	1612237	-10.80 %
L2 accesses	359574	-17.29 %
instructions	573441163	-46.42 %

v11: datastructure augmentation & short-circuiting

```
static double mortality_male[MAXAGE + 2] = {
  0.00744, /* ... */, 0.32834, 0.34935, 0.37116, 1.0
};

static void barwert(...) {
    /* ... */
    if (x>100) {
        *abw=vn_irate[1]; *rbw=1; *ebw=0; return;
    }
    if (x+n>102) {
        n = 102-x;
    }
    /* ... */
        qx_values[MAXAGE+1] = 1.0 - addrisk;
    /* ... */
}
```

v11: datastructure augmentation & short-circuiting

execution time	110ms	-7.56 %
tsc	308689569	-4.34 %
branch mispred.	1555522	-3.52 %
L2 accesses	360535	+0.27 %
instructions	552480281	-3.66 %

v12: specialization / strength reduction

```
#include <math.h>
/* ... */ pow (v, p) /* ...*/
```

v12: specialization / strength reduction

```
#include <math.h>
/* ... */ pow (v, p) /* ...*/
```

```
static double _pow(double v, int p) {
    switch(p) {
        case 0 : return 1; /* not used but speeds up things! */
        case 1 : return v;
        case 2 : return v * v;
        case 3 : return v * v * v;
        default: return exp(p * log(v));
    }
}
```

v12: specialization / strength reduction statistics

execution time	100ms	-9.09 %
tsc	284195588	-7.93 %
branch mispred.	1600785	+2.91 %
L2 accesses	360804	+0.07 %
instructions	554235508	+0.32 %

v13: loop peeling

```
p = 1;    ret_abw = 0;    ret_rbw = 0;
for (i = 0; i < n; i++) {
    q = qx_values[x + i] + addrisk;
    vn1 = vn_irate[i + 1];
    ret_abw += p * q * vn1;
    ret_rbw += p * vn0;
    p *= (1 - q);
    vn0 = vn1;
}
```

v13: loop peeling

```
p = 1;     ret_abw = 0;     ret_rbw = 0;
for (i = 0; i < n; i++) {
    q = qx_values[x + i] + addrisk;
    vn1 = vn_irate[i + 1];
    ret_abw += p * q * vn1;
    ret_rbw += p * vn0;
    p *= (1 - q);
    vn0 = vn1;
}
```

```
/* loop peeled for i=0 */
q = qx_values[x] + addrisk;      vn0 = vn_irate[1];
ret_abw = q * vn0;               ret_rbw = 1;
p = (1 - q);
for (i = 1; i < n; i++) { /* loop starting at i=1 */
    /* same as above */
}
```

v13: loop peeling — statistics

execution time	99ms	-1.00 %
tsc	276666435	-2.65 %
branch mispred.	1503056	-6.11 %
L2 accesses	361493	+0.19 %
instructions	538007567	-2.93 %

v14: sw pipelining

```
for (i = 0; i < n; i++) {
    q = qx_values[x + i] + addrisk;
    vn1 = vn_irate[i + 1];
    ret_abw += p * q * vn1;
    ret_rbw += p * vn0;
    p *= (1 - q);
    vn0 = vn1;
}
```

v14: sw pipelining

```
for (i = 0; i < n; i++) {
    q = qx_values[x + i] + addrisk;
    vn1 = vn_irate[i + 1];
    ret_abw += p * q * vn1;
    ret_rbw += p * vn0;
    p *= (1 - q);
    vn0 = vn1;
}
```

```
q = qx_values[x + 1] + addrisk;
for (i = 0; i < n; i++) {
    vn1 = vn_irate[i + 1];
    ret_abw += p * q * vn1;
    ret_rbw += p * vn0;
    p *= (1 - q);
    vn0 = vn1;
    q = qx_values[x + i + 1] + addrisk;
}
```

execution time	100ms	+1.01 %
tsc	263524010	-4.75 %
branch mispred.	1500842	-0.15 %
L2 accesses	363920	+0.67 %
instructions	545050651	+1.31 %

Surprises

Counter-optimizations

- Loop unrolling, Transfer-driven loop unrolling
- SIMD via builtin instructions
- Aggressive specialization (Code generation)
- Strength reduction for induction variables
- Profiling info for gcc

Win32 Results very sensitive to compiler options and optimizations

Profiling Optimizations often more impact as suspected

Documentation & source in LVA directory

`http://www.complang.tuwien.ac.at/anton/lvas/effizienz-abgaben/2011w/bjs`

or

`/nfs/unsafe/httpd/ftp/pub/anton/lvas/effizienz-abgaben/2011w/bjs/`

Documentation & source in LVA directory

`http://www.complang.tuwien.ac.at/anton/lvas/effizienz-abgaben/2011w/bjs`

or

`/nfs/unsafe/httpd/ftp/pub/anton/lvas/effizienz-abgaben/2011w/bjs/`

Questions?

Documentation & source in LVA directory

`http://www.complang.tuwien.ac.at/anton/lvas/effizienz-abgaben/2011w/bjs`

or

`/nfs/unsafe/httpd/ftp/pub/anton/lvas/effizienz-abgaben/2011w/bjs/`

Questions?

Thanks for your attention.