simCIR

Ein Programm zur Erzeugung von Simulationsszenarien nach dem Modell von Cox-Ingersoll-Ross

R. Stanowsky
stano [att] loop [dott] de

2013-01-08

Das Programm `simCIR` erzeugt Simulationsszenarien mehrerer Größen mit jeweils eigenem stochastischen Prozess nach dem Modell von Cox-Ingersoll-Ross. Neben Zufallsveränderungen, deren Intensität mit dem erreichten Niveau unterproportional steigt, unterstellt das Modell für jede Größe eine dem Abstand proportionale Drift hin zu einem langfristigen Mittel. Wahlweise können einzelne Größen nach einer einfacheren Formel ohne Drift und mit konstanter Zufallsintensität simuliert werden.

Die originalen Differentialgleichungen des Modells werden zur Simulation durch einfache Differenzengleichungen mit normalverteilten Zufallsschwankungen und einer gemeinsamen Kovarianzmatrix angenähert. Das Programm liest alle Parameter aus der Standardeingabe und schreibt die erzeugten Szenarien auf die Standardausgabe; Als Zufallsquelle dient /dev/urandom.

Typische Anwendung ist die Erzeugung einer Serie von langfristigen Kapitalmarktszenarien als Basis weiterer Analysen. Zinsen werden dabei mit Drift, Wertindizes für Aktien oder Immobilien ohne Drift modelliert. Das Modell betont dabei nicht kurzfristige Realitätsnähe, sondern liefert ein breites Spektrum möglicher Entwicklungen. Es eignet sich damit primär für Risikoanalysen.

Inhaltsverzeichnis

Verzeichnis der Programmteile

Verzeichnis wesentlicher Objekte

1. Programmfunktion

Nach dem Modell von Cox, Ingersoll und Ross für die Renditen festverzinslicher Titel genügen die simulierten Größen R_i mit langfristigen Mittelwerten $\dot{\mu}_i$, Driftraten $\dot{\kappa}_i$ und Volatilitätsgrößen $\dot{\sigma}_i$ stochastischen Differentialgleichungen:[1]

$$dR_i = \dot{\kappa}_i \left(\dot{\mu}_i - R_i\right) dt + \dot{\sigma}_i^2 \sqrt{R_i}\, dW_t \quad \text{mit Wienerprozess } W_t \quad i = 0 \ldots n-1$$

Für eine kleine Schrittweite δ zwischen den Punkten $t = 0, 1 \ldots T$ entspricht dies näherungsweise Differenzengleichungen mit normalverteilten Zufallsgrößen:[2]

$$R_{i,t} = R_{i,t-1} + \kappa_i \left(\mu_i - R_{i,t-1}\right) \delta + \sigma_i^2 \sqrt{R_{i,t-1}\, \delta}\, X_i \quad \text{mit} \quad X_i \sim \mathcal{N}(0,1)$$

In der Differenzengleichung werden die R_i im Unterschied zur Differentialgleichung mit geringer Wahrscheinlichkeit negativ. Dann bricht die Weiterrechnung aufgrund des Wurzelausdrucks ab und das begonnene, gemeinsame Simulationsszenario aller R_i wird verworfen und durch ein neues ersetzt.[3] Werden sinnvollerweise nicht Kassazinsen verschiedener Laufzeiten sondern (ihre impliziten) Terminzinsen von Laufzeit zu Laufzeit modelliert, so entspricht das Kriterium $R_i > 0$ der Forderung nach Arbitragefreiheit über die Zinsstruktur.[4]

Eine beliebige Auswahl der Größen R_i kann alternativ nach einer vereinfachten Differenzengleichung ohne Drift modelliert werden:

$$R_{i,t} = R_{i,t-1} + \mu_i\, \delta + \sigma_i^2 \sqrt{\delta}\, X_i \quad \text{ebenfalls mit} \quad X_i \sim \mathcal{N}(0,1)$$

Dies ist sinnvoll zur Modellierung der Logarithmen langfristig wachsender Größen ohne klare Drift zu einem bestimmten mittleren Wachstum, wie etwa von Aktien- oder Immobilienwertindizes.

Jede simulierte Größe R_i folgt hier nach einer der beiden Differenzengleichungen einem eigenen Prozess; Die standardnormalverteilten Zufallsgrößen X_i sind jedoch über eine gemeinsame Kovarianzmatrix $Cov_{n\times n}$ gekoppelt.[5] Die Simulation aller R_i wird jeweils mit den festen Startwerten $R_{i,0}$ und in gemeinsamen Schritten der Weite δ für $t = 1 \ldots T$ wiederholt bis eine vorgegebene Anzahl von Szenarien erreicht ist.

---

[1] → http://de.wikipedia.org/wiki/Cox-Ingersoll-Ross, .../Wurzel-Diffusionsprozess, .../Stochastische_Differentialgleichung, .../Wiener-Prozess

[2] → http://de.wikipedia.org/wiki/Normalverteilung

[3] Diese nachgelagerte Selektion verzerrt geringfügig die Verteilungen der X_i weg von Standardnormalverteilungen näher an die eigentlich für die fehlerfreie Überführung der Differentialgleichung in eine Differenzengleichung korrekten, nur näherungsweise normalverteilten, verschobenen und nichtzentralen Chi-Quadrat-Verteilungen. → http://de.wikipedia.org/wiki/Chi-Quadrat-Verteilung

[4] → http://de.wikipedia.org/wiki/Kassazins, .../Terminzins, .../Zinsstruktur

[5] Da die X_i die Varianz 1 besitzen ist ihre Kovarianzmatrix insbesondere auch ihre Korrelationsmatrix. Die individuelle Volatilität der Zufallsgrößen für R_i wird hier durch die σ_i^2 abgebildet. Rechnerisch könnten statt der $X_i \sim \mathcal{N}(0,1)$ auch $\tilde{X}_i = \sigma_i X_i \sim \mathcal{N}(0, \sigma_i^2)$ verwendet werden; Damit entfielen in den Gleichungen die Faktoren σ_i^2 und stattdessen wären die i-te Zeile und die i-te Spalte der Korrelations-/Kovarianzmatrix jeweils mit σ_i zu multiplizieren. Das Programm lässt jede Aufteilung der Volatilitäten auf die Faktoren σ_i^2 und die gemeinsame Kovarianzmatrix zu, sodass sie allgemein als „Kovarianzmatrix“ bezeichnet wird. Übersichtlicher und leichter interpretierbar ist jedoch die übliche Darstellung der Volatilitäten mit den Faktoren σ_i^2 und einer Korrelationsmatrix.
→ http://de.wikipedia.org/wiki/Kovarianzmatrix, .../Korrelation

2. Programmstruktur

Der Aufbau des in C geschriebenen Programms ist einfach. Mit n_MAX und T_MAX wird die maximale Anzahl n der Simulationsgrößen R_i beziehungsweise die maximale Anzahl T der Simulationsschritte festgelegt die das Programm verarbeiten kann. Das Programm akzeptiert weder Optionen noch Argumente; Den Versuch quittiert es mit einer Kurzhilfe und Versionsmeldung.

5a ⟨*simCIR.h* 5a⟩≡ (5b) 9a▷

```
int main(int argc, char *argv[]);
```

Benutzt argc 5b und argv 5b.

5b ⟨*simCIR.c* 5b⟩≡

```
⟨Geladene Bibliotheken 12c⟩
#define n_MAX 10
#define T_MAX 10001
⟨simCIR.h 5a⟩
int main(int argc, char *argv[])
{
    unsigned n = 0,
             T;
    float delta,
          mu[n_MAX-1],
          kappa[n_MAX-1],
          sigma2[n_MAX-1],
          R[n_MAX-1][T_MAX],
          Cov[n_MAX-1][n_MAX-1];
    ⟨Definition weiterer Variablen 6a⟩
    if (argc != 1)
    {
        ⟨Kurzhilfe geben 12b⟩
        return -1;
    }
    ⟨Parameter einlesen 11a⟩
    ⟨Rechnung initialisieren 11c⟩
    ⟨Cholesky-Matrix berechnen 10a⟩
    ⟨Simulationen erzeugen 6b⟩
    ⟨Aufräumen 12a⟩
    return 0;
}
⟨Permanente Unterprogrammvariablen 9b⟩
⟨BoxMuller Unterprogramm 9c⟩
```

Definiert:

argc, benutzt in Teilen 5a und 19.
argv, benutzt in Teilen 5a und 19.
Cov, benutzt in Teilen 10, 11a, und 14.
delta, benutzt in Teilen 7c, 11a, 14, und 15.
kappa, benutzt in Teilen 7c, 11a, 14, und 15.
mu, benutzt in Teilen 7c, 11a, 14, und 15.
n, benutzt in Teilen 6b, 8, 10–12, und 15.
n_MAX, benutzt in Teilen 6c und 7a.
R, benutzt in Teilen 7c, 8, 11a, 12b, 14, 15, 17, und 19.
sigma2, benutzt in Teilen 7c, 11a, 14, und 15.
T, benutzt in Teilen 6b, 8, 11a, 14, und 15.
T_MAX, nirgends sonst benutzt.

Der wesentliche, wiederholt durchlaufene Anteil der Rechenarbeit geschieht im Block ⟨*Simulationen erzeugen* 6b⟩. Der Anwender wählt mit `scnFirst` und `scnLast` die erste und die letzte Nummer der in der Ausgabe fortlaufend durchnumerierten Simulationsszenarien.Damit lassen sich die Ausgaben verschiedener Programmläufe leicht verwalten.

6a ⟨*Definition weiterer Variablen* 6a⟩≡ (5b) 6c▷
```
int scnFirst, scnLast;
```
Definiert:
`scnFirst`, benutzt in Teilen 6b, 11a, und 14.
`scnLast`, benutzt in Teilen 6b, 11a, und 14.

6b ⟨*Simulationen erzeugen* 6b⟩≡ (5b)
```
for (int scenario = scnFirst; scenario <= scnLast; scenario++)
{
newScenario:
    for (unsigned t = 1; t <= T; t++)
    {
        for (int i = 0; i < n; i++)
        {
            ⟨Neue Zufallszahl g[i] ziehen 6d⟩
            ⟨Zufallszahl x[i] kombinieren 7b⟩
            ⟨Schritt t im Prozess i machen 7c⟩
        };
    };
    ⟨Zufallsszenario ausgeben 8⟩
};
```
Definiert:
`newScenario`, benutzt in Teil 7c.
`scenario`, benutzt in Teil 8.
`t`, benutzt in Teilen 7c, 8, und 19.
Benutzt `n` 5b, `scnFirst` 6a, `scnLast` 6a, und `T` 5b.

3. Eine neue Zufallszahl g_i ziehen

Eine neue, unabhängig standardnormalverteilte Zufallszahl g_i liefert das Unterprogramm `BoxMuller`.

6c ⟨*Definition weiterer Variablen* 6a⟩+≡ (5b) ◁6a 7a▷
```
float g[n_MAX-1];
```
Definiert:
`g`, benutzt in Teilen 6d und 7b.
Benutzt `n_MAX` 5b.

6d ⟨*Neue Zufallszahl g[i] ziehen* 6d⟩≡ (6b)
```
if ( BoxMuller(fdUrandom, &g[i]) )
{
    perror("simCIR: BoxMuller subprogramme couldn't read "\
        "(enough) raw random numbers from /dev/urandom");
    return -3;
}
```

Benutzt `BoxMuller` 9c, `fdUrandom` 11b, und `g` 6c.

4. Die nächste Zufallszahl x_i kombinieren

In jedem Simulationsschritt werden n Realisationen x_i von Zufallsvariablen $X_i \sim \mathcal{N}(0,1)$ mit gemeinsamer Kovarianzmatrix $(Cov_{i,j})$ benötigt. Die x_i werden sukzessive für $i = 0$ bis $i = n-1$ aus jeweils den ersten $i+1$ von n Ziehungen g_i stochastisch unabhängiger Zufallsvariablen $G_i \sim \mathcal{N}(0,1)$ linear kombiniert. Mit den Spaltenvektoren $\vec{G} = (G_i)$ und $\vec{X} = (X_i)$ sowie der Cholesky-Matrix[6] $\mathcal{C}_{n\times n} = (c_{i,j})$ schreibt sich dies:

$$\vec{X} = \begin{pmatrix} X_0 \\ \vdots \\ X_{n-1} \end{pmatrix} = \begin{pmatrix} c_{0,0} & & 0 \\ \vdots & \ddots & \\ c_{n-1,0} & \cdots & c_{n-1,n-1} \end{pmatrix} \times \begin{pmatrix} G_0 \\ \vdots \\ G_{n-1} \end{pmatrix} = \mathcal{C} \times \vec{G}$$

Die x_i werden gleich nach Berechnung verbraucht und daher alle in der einen Variablen `x` berechnet.

7a ⟨*Definition weiterer Variablen* 6a⟩+≡ (5b) ◁6c 11b▷

```
float Cholesky[n_MAX-1][n_MAX-1];
```

Definiert:
`Cholesky`, benutzt in Teilen 7b und 10.
Benutzt `n_MAX` 5b.

7b ⟨*Zufallszahl x[i] kombinieren* 7b⟩≡ (6b)

```
float x = 0;
for (int j = 0; j <= i; j++)
    x += Cholesky[i][j] * g[j];
```

Benutzt `Cholesky` 7a und `g` 6c.

5. Einen einzelnen Simulationsschritt machen

Da ein $\kappa_i < 0$ für den CIR-Prozess ungeeignet ist dient die Eingabe als Signal den Prozess i nach der vereinfachten Gleichung zu erzeugen. Bei Auftreten einer $R_{i,t} < 0$ in den CIR-Prozessen wird die Szenarioberechnung neu begonnen ohne den Zähler für die Szenarien zu verändern.

7c ⟨*Schritt t im Prozess i machen* 7c⟩≡ (6b)

```
if (kappa[i] >= 0)
{
    if ( ( R[i][t] = R[i][t-1] + kappa[i] * (mu[i] - R[i][t-1]) * delta \
                + sigma2[i] * sqrt(R[i][t-1] * delta) * x ) < 0 )
        goto newScenario;
}
else
    R[i][t] = R[i][t-1] + mu[i] * delta + sigma2[i] * sqrt(delta) * x;
```

Benutzt `delta` 5b, `kappa` 5b, `mu` 5b, `newScenario` 6b, `R` 5b, `sigma2` 5b, und `t` 6b.

[6] → http://de.wikipedia.org/wiki/Cholesky-Zerlegung

6. Ein Zufallsszenario ausgeben

Jedes volle Szenario wird sofort auf die Standardausgabe geschrieben. Zur einfacheren Kontrolle werden den Simulationsschritten die Nummer des Szenarios und die Nummer des Schrittes vorangestellt. Getrennt wird mit Leerzeichen und Zeilenumbrüchen die von nachfolgenden Programmen mutmaßlich am leichtesten erkannt werden.

8 ⟨*Zufallsszenario ausgeben* 8⟩≡ (6b)

```
for (unsigned t = 0; t <= T; t++)
{
    printf("%4u %4u ", scenario, t);
    for (int j = 0; j < n; j++)
        printf("%+1.8f ", R[j][t]);
    printf("\n");
};
```

Benutzt `n` 5b, `R` 5b, `scenario` 6b, `T` 5b, und `t` 6b.

7. Standardnormalverteilte Zufallszahlen erzeugen

Aus den von /dev/urandom gelieferten ganzzahligen (Pseudo-)Zufallszahlen sind zwei auf dem Intervall $(0,1)$ (näherungsweise) unabhängig gleichverteilte Zufallszahlen u_1, u_2 leicht zu gewinnen.[7] Daraus werden nach der Box-Muller-Methode zwei Zufallsgrößen z_1, z_2 berechnet:[8]

$$z_1 = r\cos\varphi,\ z_2 = r\sin\varphi \quad \text{mit} \quad r = \sqrt{-2\ln u_1},\ \varphi = 2\pi u_2$$

Mit der Rechteckverteilung $Re(0,1)$ auf dem Intervall $(0,1)$ sowie der Exponentialverteilung $Ex(\lambda)$ mit Dichte $\lambda e^{-\lambda x}$ und Verteilungsfunktion $1-e^{\lambda x}$ gilt:[9]

$$e^{-\frac{1}{2}r^2} = u_1 \sim Re(0,1) \Leftrightarrow 1 - e^{-\frac{1}{2}r^2} \sim Re(0,1) \Leftrightarrow r^2 \sim Ex(\frac{1}{2})$$

Weiter ist φ auf dem Intervall $(0,2\pi)$ gleichverteilt mit zugehöriger Dichte $\frac{1}{2\pi}$. Für die gemeinsame Verteilung gilt auf der Halbachse $\varphi = 0$, $r > 0$:

$$\frac{1}{2}e^{-\frac{1}{2}r^2}\cdot\frac{1}{2\pi}\,dr^2\,d\varphi \overset{\varphi=0,r>0}{=} \frac{1}{4\pi}e^{-\frac{1}{2}r^2}\underbrace{\frac{dr^2}{dr}}_{2r}\underbrace{\frac{dr}{dz_1}}_{1}dz_1\underbrace{\frac{d\varphi}{dz_2}}_{1/r}dz_2$$

$$\overset{r^2=z_1^2+z_2^2}{=} \frac{1}{2\pi}e^{-\frac{1}{2}(z_1^2+z_2^2)}\,dz_1\,dz_2 = \frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z_1^2}\cdot\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z_2^2}\,dz_1\,dz_2$$

Aus der Gleichheit auf der Halbachse folgt mit der Rotationssymmetrie der Produktdichte um den Nullpunkt die Gleichheit der Verteilungen in der Ebene. Die erzeugten z_1, z_2 sind mit ihrer zugehörigen Dichte $\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}x^2}$ somit (näherungsweise) unabhängig standardnormalverteilt.[10]

[7] → http://de.wikipedia.org/wiki//dev/random. Je nach Systemumgebung wird es anderen Zugang zu solchen Zufallszahlen geben.
[8] → http://de.wikipedia.org/wiki/Box-Muller-Methode
[9] → http://de.wikipedia.org/wiki/Stetige_Gleichverteilung, .../Exponentialverteilung
[10] → http://de.wikipedia.org/wiki/Normalverteilung, .../Mehrdimensionale_Normalverteilung

9a ⟨*simCIR.h* 5a⟩+≡ (5b) ◁5a

```
int BoxMuller(int fd, float *z);
```

Benutzt BoxMuller 9c.

9b ⟨*Permanente Unterprogrammvariablen* 9b⟩≡ (5b)

```
int needMoreGauss = 1;
const double twoPi = 2.0 * 3.14159265358979323846;
double r, phi;
```

9c ⟨BoxMuller *Unterprogramm* 9c⟩≡ (5b)

```
int BoxMuller(int fd, float *z)
{
    if ( needMoreGauss )
    {
        unsigned long u1, u2;

        if ( read(fd, &u1, sizeof(u1)) != sizeof(u1) )
            return -3;
        if ( read(fd, &u2, sizeof(u2)) != sizeof(u2) )
            return -3;
        r = sqrt(-2.0 * log((u1 + 0.5) / (UINT_MAX + 1.0)));
        phi = twoPi * (u2 + 0.25) / UINT_MAX;
        *z = (float) r * cos(phi);
        needMoreGauss = 0;
    }
    else
    {
        *z = (float) r * sin(phi);
        needMoreGauss = 1;
    };
    return 0;
}
```

Definiert:
BoxMuller, benutzt in Teilen 6d und 9a.

Die Berechnung von r meidet $\ln(0)$ und $\ln(1)$; Letzteres um ein gehäuftes Ziehen der 0 zu verhindern. Bei der Berechnung von φ wird durch Addition von 0.25/UINT_MAX die Symmetrie der gezogenen $\cos\varphi, \sin\varphi$ zu den Koordinatenachsen gestört um die Verteilung zu verfeinern. Wegen der paarweisen Erzeugung der z_1, z_2 wird bei jedem zweiten Aufruf wenig gerechnet.

8. Die Cholesky-Matrix berechnen

Die untere Dreiecksmatrix $\mathcal{C}$ wird vor den Simulationen einmalig, und zwar spaltenweise von links nach rechts mit jeweils den Diagonalelementen zuerst, berechnet.

10a ⟨*Cholesky-Matrix berechnen* 10a⟩≡ (5b)

```
for (int j = 0; j < n; j++)
{
    ⟨Diagonalelement C[j][j] berechnen 10b⟩
    for (int i = j+1; i < n; i++)
    {
        ⟨Spaltenelement C[i][j] berechnen 10c⟩
    };
};
```

Benutzt `n` 5b.

Ihre Elemente $c_{i,j}$ kombinieren die x_i aus den gezogenen g_i linear und auf einfache Weise gerade so, dass die Verteilung der X_i die vorgegebene Kovarianzmatrix Cov besitzt. Die Diagonalelemente lassen sich über

$$\begin{aligned} Cov_{j,j} &= Cov(X_j, X_j) = Cov\left(\sum_{k=0}^{j} c_{j,k}G_k, \sum_{k=0}^{j} c_{j,k}G_k\right) = \sum_{k=0}^{j} c_{j,k}^2 \\ \overset{\text{da } c_{j,j}>0}{\Longleftrightarrow} c_{j,j} &= +\sqrt{Cov_{j,j} - \sum_{k=0}^{j-1} c_{j,k}^2} \end{aligned}$$

leicht ermitteln:

10b ⟨*Diagonalelement C[j][j] berechnen* 10b⟩≡ (10a)

```
float f = Cov[j][j];
for (int k = 0; k < j; k++)
    f -= Cholesky[j][k] * Cholesky[j][k];
Cholesky[j][j] = sqrt(f);
```

Benutzt `Cholesky` 7a und `Cov` 5b.

Anschließend lassen sich auch die restlichen Spaltenelemente mit

$$\begin{aligned} Cov_{i,j} &= Cov(X_i, X_j) = Cov\left(\sum_{k=0}^{i} c_{i,k}G_k, \sum_{k=0}^{j} c_{j,k}G_k\right) = \sum_{k=0}^{j} c_{i,k}c_{j,k} \\ \overset{\text{da } c_{j,j}>0}{\Longleftrightarrow} c_{i,j} &= \left(Cov_{i,j} - \sum_{k=0}^{j-1} c_{i,k}c_{j,k}\right)/c_{j,j} \end{aligned}$$

einfach berechnen:

10c ⟨*Spaltenelement C[i][j] berechnen* 10c⟩≡ (10a)

```
float f = Cov[i][j];
for (int k = 0; k < j; k++)
    f -= Cholesky[i][k] * Cholesky[j][k];
Cholesky[i][j] = f / Cholesky[j][j];
```

Benutzt `Cholesky` 7a und `Cov` 5b.

9. Simulationsparameter einlesen

Die Simulationsparameter werden von der Standardeingabe gelesen. Ihre Reihenfolge einschließlich der Schlüsselwörter „scnFirst:=“ ... ist fest vorgegeben; Abweichungen führen zu undefinierten Ergebnissen. Die als Trennzeichen verwendeten Leerzeichen und Zeilenumbrüche dürfen jeweils aus beliebig vielen Leerzeichen, Zeilenumbrüchen und Tabulatorzeichen bestehen. Dateiinhalt nach der letzten einzulesenden Kovarianzgröße wird ignoriert und kann der Kommentierung dienen. Anhang A gibt ein Beispiel.

11a ⟨*Parameter einlesen* 11a⟩≡ (5b)

```
scanf(" scnFirst := %i\n", &scnFirst);
scanf(" scnLast := %i\n", &scnLast);
scanf(" T := %u\n", &T);
scanf(" delta := %f\n", &delta);

scanf(" mu := %f", &mu[0]);
while ( scanf(" %f", &mu[++n]) == 1 );

scanf(" kappa := %f", &kappa[0]);
for (int i = 1; i < n; i++)
    scanf(" %f", &kappa[i]);

scanf(" sigma2 := %f", &sigma2[0]);
for (int i = 1; i < n; i++)
    scanf(" %f", &sigma2[i]);

scanf(" R[.][0] := %f", &R[0][0]);
for (int i = 1; i < n; i++)
    scanf(" %f", &R[i][0]);

scanf(" Cov := %f", &Cov[0][0]);
for (int i = 1; i < n; i++)
    for (int j = 0; j <= i; j++)
        scanf(" %f", &Cov[i][j]);
```

Benutzt `Cov` 5b, `delta` 5b, `kappa` 5b, `mu` 5b, `n` 5b, `R` 5b, `scnFirst` 6a, `scnLast` 6a, `sigma2` 5b, und `T` 5b.

10. Rechnung initialisieren

Vor Beginn der Simulationsrechnungen muss /dev/urandom zum Lesen geöffnet werden; Ein etwaiger Fehlerrückgabewert wird durchgereicht.

11b ⟨*Definition weiterer Variablen* 6a⟩+≡ (5b) ◁7a

```
int fdUrandom;
```

Definiert:
`fdUrandom`, benutzt in Teilen 6d, 11c, und 12a.

11c ⟨*Rechnung initialisieren* 11c⟩≡ (5b)

```
if ( (fdUrandom = open("/dev/urandom", O_RDONLY) ) == -1 )
{
    perror("simCIR couldn't open /dev/urandom");
    return -2;
};
```

Benutzt `fdUrandom` 11b.

11. Aufräumen

Außer /dev/urandom wieder zu schließen bleibt am Ende nichts aufzuräumen. Ein etwaiger Fehlerwert wird durchgereicht.

12a ⟨*Aufräumen* 12a⟩≡ (5b)

```
if ( close(fdUrandom) )
{
    perror("simCIR couldn't close /dev/urandom");
    return -4;
};
```

Benutzt `fdUrandom` 11b.

12. Kurzhilfe geben

12b ⟨*Kurzhilfe geben* 12b⟩≡ (5b)

```
fprintf(stderr, "\nThis is programme simCIR v0.96 as of 2013-01-08.\
        \nCorrect call is with neither options nor arguments.\
        \nInput is read from stdin, output written to stdout,\
        \nerror messages to stderr.\
        \nFor short help check the man-page.\
        \nThe noweb file ""simCIR.nw"" including full documentation\
        \nis available at <http://sfd.koelnerlinuxtreffen.de>.\
        \nCopyright (C), 2012, R. Stanowsky; published under GPL V. 3.\
        \nstano [att] loop [dott] de\n");
```

Benutzt `n` 5b und `R` 5b.

13. Benötigte C-Bibliotheken

Um Standard-Ein-/Ausgabe, /dev/urandom, mathematische Funktionen und `MAX_UINT` nutzen zu können werden C-Standardbibliotheken geladen.

12c ⟨*Geladene Bibliotheken* 12c⟩≡ (5b)

```
#include <fcntl.h>      /* open(), O_RDONLY */
#include <limits.h>     /* MAX_UINT */
#include <math.h>       /* sqrt(), log(), sin(), cos() */
#include <stdio.h>      /* scanf(), printf(), perror() */
#include <unistd.h>     /* read(), close(), stdin, stdout */
```

14. Schlussbemerkungen

Um das Programm einfach zu halten wurden einige Kompromisse gemacht:

- Die Größe der Vektoren im C-Code ist statisch vorgegeben und begrenzt die mögliche Anzahl zu simulierender Größen sowie die maximale Anzahl zu simulierender Schritte pro Szenario.
- Die Eingabedaten müssen in einem starrem Schema arrangiert sein. Insbesondere wäre schön, die Kovarianzen würden nicht nur als untere Dreiecksmatrix sondern wahlweise auch als obere Dreiecksmatrix akzeptiert.
- Eine Funktion zum Schätzen der Simulationsparameter direkt aus empirischen Zeitreihen wäre auch praktisch.
- Wie beschrieben werden die originalen Cox-Ingersoll-Ross Differentialgleichungen durch Differenzengleichungen mit $X_i \sim \mathcal{N}(0,1)$ näherungsweise nachgebildet. Mit verschobenen, nicht-zentral Chi-Quadrat-verteilten X_i könnte der Übergang genau sein. Die Erzeugung solcher Zufallszahlen wäre jedoch erheblich aufwendiger.
- Die Box-Muller-Methode zur Erzeugung standardnormalverteilter Zufallszahlen ist zwar relativ langsam, aber leicht und zuverlässig zu implementieren.
- Die Qualität der Simulation hängt wesentlich von der Qualität der verwendeten Zufallszahlen ab. Hier wird einfach /dev/urandom vertraut.

Andererseits ist einiger Mehraufwand getrieben um das Programm ohne Aufrufe von Funktionen aus Spezialbibliotheken zu realisieren.

- Die ansonsten vorzügliche GSL/GNU Scientific Library[11] bietet zwar geeignete Funktionen (sogar zur Erzeugung Chi-Quadrat-verteilter (Pseudo-)Zufallszahlen), jedoch sind die Funktionen bis auf Hinweise auf zumeist unfreie wissenschaftliche Literatur nahezu unkommentiert und ohne Konsultation der angegebenen Literatur kaum nachzuvollziehen.
- Ein freies, umfassend dokumentiertes Programm mit Aufrufen in eine zwar freie aber nur unfrei dokumentierte Bibliotheksfunktion zu schreiben wäre unbefriedigend gewesen. Global gesehen hätte es die Auswahl der Menschen mit einer Chance das Programm vollständig nachzuvollziehen erheblich eingeschränkt. Vorausgesetzt würden der Zugang zu einer das Fachbuch/die Fachzeitschrift führenden Bibliothek oder manchmal wahlweise zu einem Online-Zahlungssystem um den Artikel herunterzuladen und jeweils die finanziellen Mittel hierzu.
- Ferner müsste das eigene Programm den Konventionen der Bibliothek angepasst werden und ein Teil der eigenen Dokumentation müsste sich hiermit statt mit der Erläuterung des eigentlich interessanten Rechenweges befassen.
- Durch die nun erreichte vollkommen selbständige Programmierung und genau passende Dokumentation sollte das Programm ohnehin am schnellsten vollständig zu verstehen sein.

[11] → http://www.gnu.org/software/gsl/

A. Eingabebeispiel

Folgende Eingabedaten könnten etwa dazu dienen in zehn gemeinsamen Szenarien für einen Zeitraum von 110 Jahren die Entwicklung von fünf (Termin-)Zinsen für unterschiedliche Laufzeiten nach dem Modell von Cox-Ingersoll-Ross sowie der Logarithmen je eines Aktien- und eines Immobilienwertindex nach der vereinfachten Gleichung in Monatsschrittweiten mit annualisierten Parametern zu simulieren.

14 ⟨*simCIRin.txt* 14⟩≡

```
scnFirst:= 1
scnLast:= 10
T:= 1320
delta:= 0.083333333333333
mu:= 0.01 0.02 0.04 0.05 0.05 0.06 0.05
kappa:= 0.2 0.2 0.1 0.1 0.5 -1 -1
sigma2:= 0.03 0.04 0.04 0.03 0.03 0.2 0.025
R[.][0]:= 0.005 0.015 0.03 0.045 0.035 0 0
Cov:= 1.00
      0.60  1.00
      0.30  0.70  1.00
      0.00  0.40  0.60  1.00
     -0.05  0.25  0.60  0.80  1.00
      0.10  0.30  0.10 -0.05  0.20  1.00
      0.00  0.00  0.00  0.00  0.00  0.00  1.00

Example input data for Cox-Ingersoll-Ross simulation
as of: 2012-10-28
source: simCIR v0.95 example

The noweb file "simCIR.nw" v0.96 as of 2013-01-08 including this example
and full documentation is available at <http://sfd.koelnerlinuxtreffen.de>.
Copyright (C), 2012, R. Stanowsky; published under GPL V. 3.
stano [att] loop [dott] de
```

Benutzt `Cov` 5b, `delta` 5b, `kappa` 5b, `mu` 5b, `R` 5b, `scnFirst` 6a, `scnLast` 6a, `sigma2` 5b, und `T` 5b.

B. Beispiel-Kontroll-Rechnung mit R

Aus den von **simCIR** erzeugten Simulationsszenarien können die Realisationen x_i der Zufallsgrößen X_i zurück gerechnet werden über

$$X_i = (R_{i,t} + (\kappa_i\,\delta - 1)R_{i,t-1} - \kappa_i\,\mu_i\,\delta)/(\sigma_i^2\sqrt{R_{i,t-1}\,\delta})$$

beziehungsweise

$$X_i = (R_{i,t} - R_{i,t-1} - \mu_i\,\delta)/(\sigma_i^2\sqrt{\delta}) \quad .$$

Damit kann die empirische Verteilung der x_i zum Beispiel mit R[12] untersucht werden.

In der schließlich generierten Graphik stellt die breitere, gelbe Linie die Verteilungsfunktion der Standardnormalverteilung dar und die schmalere, schwarze die empirische Verteilungsfunktion aller x_i. Eine weitgehende Übereinstimmung beider Kurven gibt damit auch einen Anhalt für die Güte der Näherung der Differential- durch die Differenzengleichung mit normalverteilten X_i, wobei gegebenfalls der Anteil der ohne Drift modellierten Größen zu beachten ist.

15 ⟨*simCIR.R* 15⟩≡

```
# The noweb file "simCIR.nw" v0.96 as of 2013-01-08 including this R file
# and full documentation is available at <http://sfd.koelnerlinuxtreffen.de>.
# Copyright (C), 2012, R. Stanowsky; published under GPL V. 3.
# stano [att] loop [dott] de
#
delta <- 0.083333333333333
mu <- c(0.01, 0.02, 0.04, 0.05, 0.05, 0.06, 0.05)
kappa <- c(0.2, 0.2, 0.1, 0.1, 0.5, -1, -1)
sigma2 <- c(0.03, 0.04, 0.04, 0.03, 0.03, 0.2, 0.025)
Rst <- read.table("simCIRout.txt")

T <- max(Rst[2])
N <- dim(Rst)
N <- N[2] - 2
R <- Rst[,-1]
R <- as.matrix(R[,-1])
R0 <- R[Rst[2]!=T,]
R1 <- R[Rst[2]!=0,]

X <- NULL
Xi <- NULL
for (i in 1:N) {
  if (kappa[i] >= 0)
    Xi <- (R1[,i] + (kappa[i] * delta - 1) * R0[,i]
      - kappa[i] * mu[i] * delta) / (sigma2[i] * sqrt(R0[,i] * delta))
  else
    Xi <- (R1[,i] - R0[,i] - mu[i] * delta) / (sigma2[i] * sqrt(delta))
  X <- cbind(X,Xi) }

library(fUtilities)
cat("column means:\n", colMeans(X))
cat("\n\ncolumn variations:\n", colVars(X))
cat("\n\ntotal mean: ", mean(X))
cat("        total variation: ", var(as.vector(X)))
cat("\n\ncorrelation matrix:\n")
```

[12] → http://www.gnu.org/software/r/

```
  print(cor(X))

  D <- NULL
  pValue <- NULL
  for (i in 1:N) {
    kst <- ks.test(X[,i], pnorm)
    D <- c(D, kst$statistic)
    pValue <- c(pValue, kst$p.value) }
  cat("\n", kst$method, "    ", kst$alternative, "\ncolumn tests:\n")
  print(rbind(D, pValue))

  plot(pnorm, -4, 4, lwd=5, col="yellow",
    xlab="yellow = pnorm    black = sample", ylab="(e)cdf")
  plot(ecdf(X), add=TRUE)
```

Benutzt `delta` 5b, `kappa` 5b, `mu` 5b, `n` 5b, `R` 5b, `sigma2` 5b, und `T` 5b.

C. makefile

Dieses `makefile` ist dazu gemacht vorab etwa mit `notangle -t -Rmakefile simCIR.nw > makefile` extrahiert zu werden.[13] Mit `make` wird dann das ausführbare Programm `simCIR` erzeugt. Außer `make` mit dem vollen Namen der zu erzeugenden Datei aufzurufen ist die Verwendung einiger Abkürzungen möglich, wie z.B. `make dvi` als Abkürzung für `make simCIR.dvi`. Insgesamt werden folgende Argumente erkannt:

Ohne Argument wird das ausführbare Programm `simCIR` erzeugt.

`h` Die C-Deklarationsdatei `simCIR.h` mit den Funktionsprototypen wird erzeugt.

`dvi|ps|pdf|html` Die Dokumentation `simCIR.dvi|ps|pdf|html` wird im gewählten Format aus der `noweb`-Datei erzeugt (Das HTML-Ergebnis ist mangelhaft).

`c` Ein C-Programm `simCIR.c` mit dem LaTeX-Dokumentations-Quellcode eingebettet als herkömmliche C-Kommentare wird erzeugt. Dies ist nur als Notlösung gedacht.

`in` Das Eingabebeispiel `simCIRin.txt` wird extrahiert.

`out` Das Ergebnis eines Laufs des ausführbaren Programms `simCIR` mit den Beispieldaten aus `simCIRin.txt` wird in die Datei `simCIRout.txt` geschrieben.

`R` Der R-Code `simCIR.R` der Beispiel-Kontrollrechnung wird extrahiert.

`man` Erzeugung der Datei `simCIR.1.gz` der (deutschen) man-Seite **simCIR**(1) im aktuellen Verzeichnis. Die Seite muss gegebenenfalls manuell in das geeignete Verzeichnis (z.B. /usr/share/man/de/man1/) verschoben werden.

`all` Alle vorgenannten Dateien werden erzeugt.

`clean` Selten benötigte Dateien (keine der vorgenannten, sondern als Zwischenschritt verwandte wie z.B. die LaTeX-Dateien) werden gelöscht.

`cleanall` Alle Dateien inklusive diesem `makefile` außer `simCIR.nw` werden gelöscht.

17 ⟨*makefile* 17⟩≡

```
# The noweb file "simCIR.nw" v0.96 as of 2013-01-08 including this makefile
# and full documentation is available at <http://sfd.koelnerlinuxtreffen.de>.
# Copyright (C), 2012, R. Stanowsky; published under GPL V. 3.
# stano [att] loop [dott] de
#
.PHONY : h c in out R dvi ps pdf html man all clean cleanall

simCIR : simCIR.o
	gcc -std=c99 -Wall -lm $< -o $@

simCIR.o : simCIRpure.c
	gcc -c -std=c99 -Wall -lm $< -o $@

simCIRpure.c : simCIR.nw
	notangle -L -RsimCIR.c $< | cpif $@

h : simCIR.h
```

[13] → http://www.gnu.org/software/make/

```
simCIR.h : simCIR.nw
        notangle -R$@ $< | cpif $@

c : simCIR.c
simCIR.c : simCIR.nw
        nountangle -c -R$@ $< | cpif $@

simCIR.tex : simCIR.nw
        noweave -delay -index $< | cpif $@

dvi : simCIR.dvi
simCIR.dvi : simCIR.tex
        latex $<
        latex $<
        latex $<

ps : simCIR.ps
simCIR.ps : simCIR.dvi
        dvips -f < $< > $@

pdf : simCIR.pdf
simCIR.pdf : simCIR.tex
        pdflatex $<
        pdflatex $<
        pdflatex $<

html : simCIR.html
simCIR.html : simCIR.nw
        noweave -filter l2h -delay -index -html $< | htmltoc | cpif $@

in : simCIRin.txt
R : simCIR.R
simCIRin.txt simCIR.R : simCIR.nw
        notangle -R$@ $< | cpif $@

out : simCIRout.txt
simCIRout.txt : simCIRin.txt simCIR
        ./simCIR < $< > $@

man : simCIR.1.gz
simCIR.1.gz : simCIR.nw
        notangle -RsimCIR.1 $< | gzip -9 | cpif $@

all : simCIR h c in out R dvi ps pdf html man

clean :
        rm -f simCIR.o simCIRpure.c simCIR.aux simCIR.toc simCIR.log \
           simCIR.out simCIR.tex

cleanall : clean
        rm -f simCIR simCIR.h simCIR.c simCIR.R simCIR.dvi simCIR.ps \
           simCIR.pdf simCIR.html simCIRin.txt simCIRout.txt simCIR.1.gz \
           makefile
```

Benutzt R 5b.

D. man-Seite

Zumindest eine kurze man-page[14] sollte jedes Programm haben.

19 ⟨*simCIR.1* 19⟩≡

```
.\" Man page for simCIR
.\"
.\" Copyright (C), 2012, R. Stanowsky
.\"
.\" You may distribute under the terms of the GNU General Public
.\" License as specified in the file COPYING that comes with the man
.\" distribution.
.\"
.TH SIMCIR 1 "2013-01-08" "0.96" "Dienstprogramme für Benutzer"
.SH NAME
simCIR \- Erzeugung von Simulationsszenarien nach Cox-Ingersoll-Ross

.SH SYNTAX
.B simCIR

\fBint main(int \fIargc\fR, \fBchar \fI*argv[]\fB)\fR

int main(int argc, char *argv[])

.SH BESCHREIBUNG
Das Programm \fBsimCIR\fR simuliert mehrere Größen nach dem Modell von
Cox-Ingersoll-Ross. Es akzeptiert weder Optionen noch Argumente; Den Versuch
quittiert es mit einer Kurzhilfe und Versionsmeldung. \fBsimCIR\fR liest von
\fIstdin\fR und schreibt auf \fIstdout\fR, Fehlermeldungen auf \fIstderr\fR.

.SH BEISPIEL
.B simCIR
.RI "< " simCIRin.txt " > " simCIRout.txt

.SH RÜCKGABEWERTE
.TP
.B " 0"
Fehlerfreie Ausführung
.TP
.B -1
Fehler durch Angabe einer Option oder eines Argumentes
.TP
.B -2
Fehler beim Öffnen von /dev/urandom zum Lesen
.TP
.B -3
Fehler beim Lesen von Zufallsbytes von /dev/urandom
.TP
.B -4
Fehler beim Schließen von /dev/urandom

.SH DATEIEN
.I simCIR.nw simCIR.c simCIR.tex simCIRin.txt simCIR.R
```

[14] → http://www.infodrom.org/projects/manpages-de/, http://kernel.org/doc/man-pages/, http://www.tldp.org/manpages/man.html, http://www.schweikhardt.net/man_page_howto.html

```
Das Programm \fBsimCIR\fR ist mit dem Literate Programming tool \fBnoweb\fR
geschrieben. Die Datei \fIsimCIR.nw\fR enthält neben dem Programm
\fIsimCIR.c\fR in C, einer ausführlichen Dokumentation \fIsimCIR.tex\fR in
LaTeX, einem Eingabebeispiel \fIsimCIRin.txt\fR und einem Beispiel
\fIsimCIR.R\fR für eine Kontrollrechnung mittels R auch ein \fImakefile\fR
und diese man-page. Für weitere Informationen wird empfohlen das
\fImakefile\fR mittels

.BR "    notangle" " -t -Rmakefile "
.IR simCIR.nw " > " makefile

zu extrahieren und mit

.B "    make"
.IR dvi " | " ps " | " pdf " | " html

die ausführliche Dokumentation im gewünschten Format zu erzeugen (Die
HTML-Variante ist schlechter lesbar). Die Original-Datei \fIsimCIR.nw\fR ist
unter der GPL V. 3 veröffentlicht auf <http://sfd.koelnerlinuxtreffen.de>.

.SH FEHLER
Fehlermeldungen, Anregungen, Aufmunterungen werden erbeten an
<stano [att] loop [dott] de>

.SH AUTOR
R. Stanowsky

Diese man-page ist Bestandteil der noweb-Datei \fIsimCIR.nw\fR
```

Benutzt `argc` 5b, `argv` 5b, `R` 5b, und `t` 6b.

E. ChangeLog/Programmgeschichte

2012-09-15 Präsentation und Verwendung einer unfertigen Version in einem Vortrag anlässlich des Software Freedom Day → http://sfd.koelnerlinuxtreffen.de

2012-10-03 Veröffentlichung einer ersten Version (ohne Versionsnummer) ebenda

2012-12-04 Veröffentlichung der Version 0.95 ebenda mit folgenden Änderungen:

- Der zweidimensionale C-Vektor `R[i][t]` deutet jetzt im Einklang mit den $R_{i,t}$ der Differenzengleichungen im linken Index i auf die Simulationsgröße und im rechten Index t auf den Simulationsschritt (statt vorher $R_{i,t}$ aber `R[t][i]`).
- Detailverbesserungen von LaTeX-Dokumentation, makefile und Kurzhilfe.
- Ergänzung um eine man-page
- Ergänzung um (Hyper-)Links
- Ergänzung um dieses ChangeLog

2013-01-08 Version 0.96 mit (Detail-)berichtigten Bemerkungen zum Verhältnis der Differenzengleichungen mit normalverteilten Zufallsgrößen zu solchen mit Chi-Quadrat-verteilten Zufallsgrößen.

F. Lizenz

G. Danke schön

Als langjähriger Nutzer, Nutznießer und Freund freier und offener Programme danke ich den Autoren und Sponsoren; Umso mehr wenn die Programme zudem nicht-kommerziell, gut dokumentiert oder sogar beides sind.

Und ich danke allen, die mich auf die Idee brachten dieses Programm mit `noweb` zu schreiben, etwas auszuprobieren oder sich die Zeit nahmen mir etwas zu erklären.

> Dies Programm folgt dem Gedanken
> mich mal ein bisschen zu bedanken.