目次

第1章

変化率と複利計算

本講義のタイトルは「動態マクロ経済学」である。時間を通じて変化するマクロ経済の分析ということだ。この章では，変化量を記述する数学について基本的な部分を整理する。

1.1 概要

経済学に限らず時間を通じて変化する対象を分析するには，変化の相対的な大きさを分析の対象にすると便利なことが多い。例えば，「5 年で年収を 500 万円増やす」という目標の達成難易度は当初年収によって異なる。計画時点の年収が 500 万円の人が年収 1000 万円になるには並々ならぬ努力と転職を伴うかもしれない。しかし，初めから 5000 万円の年収を受け取っている人が年収 5500 万円になることは，（これは私の想像に過ぎないけれども）現在の仕事の延長線上として達成できそうな気がする。同じ 500 万円増であっても，前者は 100% の年収アップ，後者は 10% アップであるから，変化率を見れば大きな違いがあることが分かるだろう。

現在の年収が 500 万円であるとしよう。もう少し現実的な目標として次のような 2 つの計画を立てて，比較する。

1. 毎年 10 万円ずつの年収増
2. 毎年 2% ずつの年収増

年収が倍の 1000 万円になるには何年かかるだろうか。1 つ目のプランは年収の増分が加法的に作用する一方で，2 つ目のプランは乗法的に作用する。最初のうちは大きな違いはないが，時間が経過するにつれて差が大きくなっていく。

以下では，簡単な数値例を用いて計算方法に習熟するとともに，数式や記号を用いた抽象的な思考に慣れよう。この章では，次のことを学ぶ。

- 時間を表す変数
- 時間変化の表示方法
- 平均変化率
- 複利計算

1.2 理論

1.2.1 時間を表す変数と時間変化する変数

2017 年の GDP が 500 兆円，2018 年の GDP が 510 兆円だったとしよう[1]。2017 年から 2018 年にかけての GDP の成長は

$$\frac{510-500}{500} = \frac{10}{500} = 0.02$$

によって計算できる。少し視点を変えてみよう。2017 年の 500 兆円と，2018 年の前年度からの成長率が 2% であることが分かっていれば，2018 年の GDP は次のようにして計算できる。

$$500 \times (1 + 0.02) = 510$$

ここでちょっとした記号を導入する。少しずつ抽象化していくので，まずは腕試し

[1] GDP とは Gross Domestic Product の略で，日本語では国内総生産と訳される。ある一定期間に国内で行われた生産活動の規模を測る指標である。詳しくは第**??**章で詳しく説明する。

だ。2017 年の GDP を GDP_{17}, 2018 年の GDP を GDP_{18} と書くことにする。

$$\mathrm{GDP}_{17} = 500, \qquad \mathrm{GDP}_{18} = 510$$

また，2017 年から 2018 年にかけての**成長率**を $g_{18/17}$ と書くと，

$$g_{18/17} = \frac{\mathrm{GDP}_{18} - \mathrm{GDP}_{17}}{\mathrm{GDP}_{17}} = \frac{\mathrm{GDP}_{18}}{\mathrm{GDP}_{17}} - 1$$

および

$$\mathrm{GDP}_{18} = \left(1 + g_{18/17}\right) \mathrm{GDP}_{17}$$

と書ける。言語的な表現に近い記法を用いると読みやすい式が書ける。一方で，少し煩わしさを感じる読者もいるかもしれない。次に進もう。なお，GDP については成長率という言葉を用いることが多いが，より一般的な**増減率**とか**変化率**と同じものである。

数学モデルを構築するときにはさらに簡略化した表記が便利である。言語的な表現から離れるので，記号が何を意味しているかを常に意識しながら読む必要が生じるものの，コンパクトな表現が数式の操作を容易にしてくれるというご利益がある。時間を表現する変数には t を用いるのが慣例である。$t = 0$ が分析の起点で，$t = 1, 2, 3, \ldots$ と数字が増えるにつれて特定の定まった間隔で時間が進行する。例えば，2010 年を分析の起点として，1 年毎の観測をもとに分析を進めるとすれば，

$$t = 0 \Leftrightarrow \text{2010 年}, \quad t = 1 \Leftrightarrow \text{2011 年}, \quad \cdots$$

のような対応関係を作ることができる。なお，整数値を取る時間軸上に構築された数理モデルを「**離散時間モデル**」と呼ぶ。

GDP に対して記号 y を使うとしよう。すると，y_0 は 2010 年の GDP，y_1 は 2011 年の GDP のように理解される。この記法を用いれば，前述の数値例は

$$y_7 = 500, \qquad y_8 = 510$$

と書ける。「2017 年の GDP は 500」というかわりに，「7 期の GDP は 500」という。特定の期を固定することなく，一般の t のままで議論を進められる方が数学的には何か

と都合がよいことが多い。単に，「t 期の GDP は y_t，$(t+1)$ 期の GDP は y_{t+1} である」のように言ったときには，$t = 10$ でも $t = 208$ でも何でもよいと考えているのである。普通は t が取る値の範囲まで示して，「$y_t, t = 0, 1, 2, \ldots,$ は t 期の GDP である」のように書く[2]。y_t のように，一定時間おきの観測値を表すデータを**時系列**とか**時系列データ**という。

問題 1.1. 分析の起点を 1951 年 ($t = 0$) とする。

1. 第 58 期 ($t = 58$) は西暦に換算すると何年か？
2. 一般の t 期は西暦何年か？ t を用いた公式を導きなさい。

Answer

注意 1.1. 分析の開始期を $t = 0$ としたが，これを $t = 1$ としてもよい。$t = 3$ とか $t = 2000$ を起点としてもあまり嬉しいことはない。本書では配列のインデックスがゼロから始まる Python を用いたので，$t = 0$ を起点とする方が使いやすい。 □

注意 1.2. Python で表形式データを扱うためのライブラリ Pandas を用いると，実際の西暦や日付等をインデックスとして利用することができるので，実データ分析に限って言えば前述のような抽象化を使わずに済む場合も多い。しかし，数理モデルを使ったシミュレーションや高度な分析を行う場合には NumPy の配列を直接操作する方が都合がよいこともあるので，ゼロから始まる整数のインデックスを使う考え方に習熟しておこう。 □

[2] ここで，$t = 0, 1, 2, \ldots$ のようにピリオドを 3 つ並べた記法は以下略という意味である，省略されている内容が文脈から読み取れるとき以外には使わない。ある特定の T が t の最大値であると考えている場合には，$t = 0, 1, 2, \ldots, T$ と書く。

問題 1.2. 1 年間の GDP ではなく四半期 GDP を分析したいとしよう。2000 年の第 1 四半期（1–3 月期）を分析の起点 $t = 0$ とする。

1. $t = 1, 2, \ldots$ と四半期 GDP$y_1, y_2, \ldots$ が表す内容を下表に書き下しなさい。

記号	意味
$t = 0$	2000 年の第 1 四半期（1–3 月期）
y_0	2000 年の第 1 四半期の四半期 GDP
$t = 1$	
y_1	
$t = 2$	
y_2	

2. $t = 10$,$t = 35$ はそれぞれ何年の第何四半期にあたるか？

Answer

注意 1.3. 問題 1.2 では暦年について 4 つの四半期を考えた。1 月始まりでない一般の会計年度を用いる場合には，例えば，「2019 年度第 3 四半期」が表しているものが何かをきちんと意識する必要がある。会計年度の始まりが 4 月であれば，日本で育った人にとっての標準的な理解は「2019 年の 10–12 月期」ということになるだろう。しかし，2018 年の 10–12 月期と表すということがあるらしい。というよりむしろ後者が国際的なスタンダードのようだ。つまり，「2019 年度」というのは「その会計年度の終了月が 2019 年にある」ということになる。 □

1.2.2 時間変化の大きさ

変数 y の，t 期から $(t+1)$ 期にかけての変化を表す基本的な方法は**差分** (difference) を取ることである。

$$\Delta y_{t+1} = y_{t+1} - y_t.$$

しかし，冒頭で述べたように Δy_{t+1} の大きさが意味する内容（容易に起こりそうな変化か，なかなか起こりそうにない変化か）は y_t の大きさによって異なる。この問題は変化率を計算することで解決できる。t 期から $(t+1)$ 期にかけての y の変化率（あるいは成長率）を g_{t+1} と書くことにしよう。これは，

$$g_{t+1} = \frac{\Delta y_{t+1}}{y_t} = \frac{y_{t+1} - y_t}{y_t} = \frac{y_{t+1}}{y_t} - 1$$

と定義される。もちろん，

$$y_{t+1} = (1 + g_{t+1}) y_t$$

である。変数 y が前節の GDP を表すとすれば，

$$g_8 = 0.02$$

となる。ここでは，g_8 は 8 期の GDP の前年度からの成長率と解釈される。なお，

$$1 + g_{t+1} = \frac{y_{t+1}}{y_t}$$

は**粗成長率**（gross growth rate）という。

注意 1.4. 記号の定義は毎回必ずチェックする必要があることに注意しよう。例えば，

$$g_t = \frac{y_{t+1} - y_t}{y_t}$$

と成長率を定義する人がいても不思議ではない。逆に言えば，**数式を使った分析を実行**

するのであれば，すべての記号の定義を述べるのが読者に対する最低限の礼儀である[3]。y だから GDP とか，t だから時間だといった思い込みでさえ邪魔である。排除しよう。

注意 1.5. ただし，定義しさえすればどんな記号を使ってもよいという訳ではない。慣例的に使われている記号から逸脱することは避けたほうがよい。例えば，GDP を g として，GDP 成長率を r にするような選択をしたら大混乱を招くだろう。

問題 1.3. ある連続する 2 期の四半期 GDP の原系列 y_t, y_{t+1} が与えられているとする[4]。この 2 期間に渡って「成長率」

$$\frac{y_{t+1} - y_t}{y_t}$$

を計算することの問題点を指摘しなさい。（ヒント：購買行動の季節変化を考える）

Answer

隣接するデータ同士で成長率を計算するにはデータから季節変動を除去しなければならない（**季節調整**という）。本講義では，季節調整済みのデータのみを扱い，この問題は解決しているものとする。季節調整法の詳細に関心のある読者は**?**を参照せよ。

1.2.3　累積成長と平均成長率

$t = 0, 1, 2, \ldots$ を期を表す変数，y_t を t 期の GDP とする。$n = 1, 2, \ldots$ に対して，n 期間の成長は次のように計算される。

$$\frac{y_{t+n} - y_t}{y_t} = \frac{y_{t+n}}{y_t} - 1$$

[3] 私が未定義の記号を無断で使ってしまっている場合には教えて下さい。訂正します。

[4] 原系列というのは変換操作を施していない生の時系列データのこと。

これは n 期間で起こる成長の大きさを測る指標だから，1 期分の成長とはスケールが異なることに注意しよう。1 期が 1 年の場合の変化率には「年率」などの便利な表現があるが，期間の長さが一般の場合にも使える一般的な日本語表現が定まっていないようなので，本書では「期間変化率」とか「期間成長率」という言葉を用いることにしよう。

例 1.1. ある y_t の水準から，年率 2% の成長が 3 年間続いたとすれば（1 年を 1 期としている），3 年後の GDP,y_{t+3}, は次のように計算できる。

$$y_{t+1} = 1.02y_t, \quad y_{t+2} = 1.02y_{t+1}, \quad y_{t+3} = 1.02y_{t+2}$$

だから，

$$y_{t+3} = 1.02^3 y_t \approx 1.061 y_t$$

である。1 年間の成長が 2% のとき，3 年間の成長は 6.1% と，およそ 3 倍の大きさになる。 □

例 1.2. 国民経済計算（GDP 統計）の四半期 GDP 速報では季節調整済み前期比が公開される。前期比の成長率が g（例えば，$g = 0.01$ とか $g = 0.005$）のとき，年率換算した GDP 成長率は次のように計算される。

$$\text{四半期 GDP 成長率（季節調整済み前期比）の年率換算値} = (1+g)^4 - 1$$

□

注意 1.6. 「1 期」が表す期間が 1 年なのか四半期なのか，あるいは 1 ヶ月なのかといった違いは数理モデルの記述には現れない。それではモデルと現実をどうやって接続するのかというと，成長率や利子率の大きさと期間の長さの関係を用いることが多い。例えば，現実の GDP 成長率が年率にして 2% であるとしよう。このとき，期間成長率が 6% であるとしてモデルをセットアップするなら，1 期 ≈ 3 年 であると考えるのが自然だろう。 □

粗成長率 y_{t+n}/y_t を次のように書き換える。

$$\begin{aligned}\frac{y_{t+n}}{y_t} &= \frac{y_{t+n}}{y_{t+n-1}} \times \frac{y_{t+n-1}}{y_{t+n-2}} \times \cdots \times \frac{y_{t+2}}{y_{t+1}} \times \frac{y_{t+1}}{y_t} \\ &= \underbrace{(1+g_{t+n}) \times (1+g_{t+n-1}) \times \cdots \times (1+g_{t+2}) \times (1+g_{t+1})}_{n \text{ 個}}.\end{aligned}$$

t 期と $t+n$ 期の間の y の**平均成長率**あるいは**平均変化率**とは，当該期間で一定の期間変化率を保って y_t から y_{t+n} に変化するとしたときの，その一定の期間変化率のことである。平均成長率 $\bar{g}$ は次の性質を持つ。

$$\frac{y_{t+n}}{y_t} = \underbrace{(1+\bar{g}) \times (1+\bar{g}) \times \cdots \times (1+\bar{g}) \times (1+\bar{g})}_{n \text{ 個}} = (1+\bar{g})^n.$$

したがって，$\bar{g}$ は次のように計算できる。

$$\bar{g} = \left(\frac{y_{t+n}}{y_t}\right)^{\frac{1}{n}} - 1$$

あるいは，隣接する 2 期の間の成長率 $g_{t+1}, g_{t+2}, \ldots, g_{t+n}$ を用いると，

$$\bar{g} = \{(1+g_{t+n})(1+g_{t+n-1}) \cdots (1+g_{t+2})(1+g_{t+1})\}^{\frac{1}{n}} - 1. \tag{1.1}$$

問題 1.4. n 個の実数を適当に選び，$i = 1, 2, \ldots, n$ でインデックス付けしたものを $x_1, x_2, \ldots, x_n$ と書こう。これらの数をすべて足した値は，$\sum$ を使って次のように書く。

$$\sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n.$$

この記法を使った次の等式を示しなさい。

$$y_{t+n} - y_t = \sum_{i=1}^{n} \Delta y_{t+i}.$$

Answer

問題 1.5. 加算記号 $\sum$ の乗算バージョンは $\prod$ である。この記号は指定されたすべての数の積を表す。

$$\prod_{i=1}^{n} x_i = x_1 x_2 \times \cdots \times x_n.$$

この記法を用いて，(1.1) の公式を書き直しなさい。

Answer

1.2.4 成長率と対数関数

$b > 0$ かつ $b \neq 1$ とする。b を底とする対数関数 $\log_b(\cdot)$ は次の性質を持つ[*5]。

1. 任意の $x, y > 0$ について，

$$\log_b xy = \log_b x + \log_b y, \qquad \log_b \frac{x}{y} = \log_b x - \log_b y$$

2. 任意の $x > 0$ と任意の実数 z について，$\log_b x^z = z \log_b x$.
3. 任意の $x > 0$ について，$b^{\log_b x} = x$.
4. $\log_b 1 = 0$.
5. $\log_b b = 1$.

[*5] 関数の独立変数を囲むカッコは省略することが多い。

底 b を変えることで生じるのは，対数関数の定数倍の違いである。

命題 1.1. $b, c > 0, b \neq 1 \neq c$ とする。任意の $x > 0$ に対して，

$$\log_b x = \log_b c \times \log_c x.$$

証明. これを示すには，

$$b^{\log_b x} = b^{\log_b c \times \log_c x}$$

を示せばよい。対数関数の性質から左辺は x である。右辺の方も x と一致することを次のようにして確認できる。

$$b^{\log_b c \times \log_c x} = \left(b^{\log_b c}\right)^{\log_c x} = c^{\log_c x} = x.$$

□

底としてよく使われるのは，$b = 2$ や 10 である。$\log_{10}(\cdot)$ を**常用対数**という。

例 1.3. 変数を 2 倍にすると $\log_2(\cdot)$ を取った値は 1 だけ大きくなる。なぜなら，

$$\log_2 2x = \log_2 2 + \log_2 x = 1 + \log_2 x.$$

同様の方法で，変数を 10 倍にすると $\log_{10}(\cdot)$ は 1 だけ大きくなることを示せる。 □

$\log_2(\cdot)$ や $\log(\cdot)$ は人間が数字を評価するときには使いやすいが，数学的に最も便利という訳ではない。微分積分と最も親和性が高いのが**自然対数** $\log_e(\cdot)$ である。底 e は**ネイピア数**や**自然対数の底**と呼ばれ，次のように定義される。

$$e = \lim_{n \to +\infty} \left(1 + \frac{1}{n}\right)^n. \tag{1.2}$$

自然対数 $\log_e(\cdot)$ は通常，$\log(\cdot)$ のように e を省略して書くか，$\ln(\cdot)$ という特別な記号を用いて表現する。この講義では $\log(\cdot)$ という記法を採用する。

自然対数の微分

$$\left(\log x\right)' = \frac{1}{x}$$

は証明なしで認めておこう[6]。正値関数 $f(\cdot)$ の自然対数の微分もよく使われるので，覚えておこう。

$$\big(\log f(x)\big)' = \frac{f'(x)}{f(x)}.$$

次の性質はよく使われる。

命題 1.2. x がゼロに十分近いとき，次の近似式が成り立つ[7]。

$$\log(1+x) \approx x$$

問題 1.6. 命題 1.2 をテイラー展開を使って証明せよ。

Answer

図 1.1 には $y = x$ と $y = \log(1+x)$ のグラフが描かれている。$x = 0$ の周りで 2 つの関数が近接していることを確認してほしい。$|x|$ が大きくなるとグラフは次第に離れていく。つまり，$|x|$ が大きいときには命題 1.2 の近似は使えなくなるなる。

この命題を用いると「**対数差分は変化率を近似する**」ことが分かる。すなわち，

命題 1.3. $y_t > 0$ と $y_{t+1} > 0$ が十分近ければ，次の近似公式が成り立つ。

$$\Delta \log y_{t+1} = \log y_{t+1} - \log y_t \approx \frac{\Delta y_{t+1}}{y_t}$$

[6] 指数関数や対数関数をどのように定義するかによって議論が変わってくる。大学初年次で使った微分積分学の教科書を参考にせよ。

[7] 「十分近い」とか「十分大きい」という数学的な表現はよく使うので意味するところを覚えておこう。ここでは $|x| \to 0$ の極限で $|\log(1+x) - x| \to 0$ が成り立つという意味である。

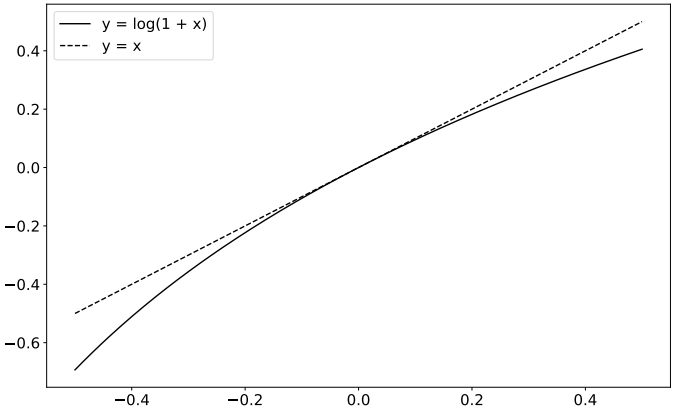

図 1.1: 対数関数の近似

証明. 対数の差が商の対数になることに注意すると,

$$\begin{aligned}\Delta \log y_{t+1} &= \log y_{t+1} - \log y_t \\ &= \log \frac{y_{t+1}}{y_t} \\ &= \log \left(1 + \frac{y_{t+1} - y_t}{y_t}\right) \\ &= \log \left(1 + \frac{\Delta y_{t+1}}{y_t}\right) \\ &\approx \frac{\Delta y_{t+1}}{y_t}.\end{aligned}$$

□

例 1.4 (Rule of 70)**.** 年率 x % で成長している変数が 2 倍になるために必要な年数はおよそ $70/x$ 年である。これは「Rule of 70」と呼ばれる近似公式である。例えば，年率 2% で成長している変数は，35 年で 2 倍になる。証明を与えておこう。当初 y である変数

が年率 x% で成長して T 年で 2 倍になるとすれば，y, x, T について次の式が成り立つ。

$$\left(1 + \frac{x}{100}\right)^T y = 2y$$

両辺の対数を取ると，

$$T \log\left(1 + \frac{x}{100}\right) + \log y = \log 2 + \log y$$

したがって，

$$T = \frac{\log 2}{\log\left(1 + \frac{x}{100}\right)} \approx \frac{\log 2}{x/100},$$

最後の近似には命題 1.3 を使っている。$\log 2 = 0.693147\cdots$ なので，

$$T \approx \frac{69.3}{x}$$

という近似が成り立つ。計算が容易になるように，$T \approx 70/x$ という公式がよく使われている。 □

1.2.5 複利計算と指数関数

指数的成長

銀行に 100 万円を預けているとする。利息の年率を 6% とする。もちろん，計算を簡単にするために選んだ数字だ。追加の預け入れも引き出しもしなければ，1 年後には 106 万円になっている（諸々の税・手数料等を支払った後の利率が 6% としている）。さらに 1 年間預金に手を付けなければ預金残高は 112.36 万円となる。前節までで扱った成長の公式と同じ計算方式なので，すでに予想はついていると思うが，t 年後 ($t = 1, 2, 3, \ldots$) の預金残高は $1.06^t \times 100$ 万円である。受け取った利息にも利息が付くので，t が増えるたび資産の増え方が早くなっていく。金融資産の評価の文脈では，このような計算方式を**複利計算**（compounding）と呼ぶ。

複利計算と似て非なるものとして単利計算という計算方式がある。これは，利息には利息がつかないという契約である。上記の例で利息を毎回引き出してタンス貯金すれ

ば，t 年後の資産残高（銀行 + タンス）は $(100 + 6t)$ 万円になる[*8]。複利計算と単利計算は短期的には大きな違いはないが，運用期間が長くなるにつれて差がどんどん大きくなる。図 1.2 で確認してほしい。

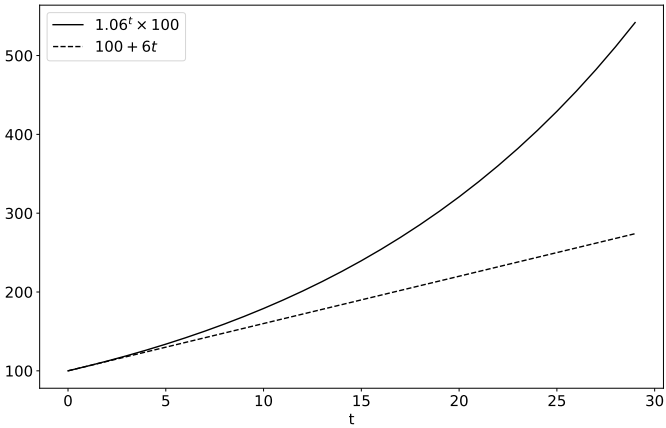

図 1.2: 複利計算と単利計算

GDP 成長率の国際比較をするときに見かけ上は小さな差（例えば，1% か 2% か）を無視できないのは，小さな成長率の格差が長期的に大きな GDP の差につながることがあるからだ。複利計算のときのようにどんどん増えていくような状況を「指数的に増える」とか「幾何級数的に増える」などと表現する。

ネイピア数再び

さて，ここまでは利息が毎年一回受け取れるものとして計算を進めてきた。もっと頻繁に利息を受け取れるとすればどうなるだろう。例えば，毎月 1 回あるいは毎日 1 回利息が支払われるとすれば 1 年後の預金残高はいくらになっているだろうか。

[*8] タンス預金にしなくても，このような状況を作ることができる。例えば，平均リターン 6% の投資信託を 100 万円分購入し，配当を再投資しないように設定すればよい。

各回の利息を計算するための基本の計算式は

$$\text{各回の利率} = \text{年率} \div \text{年間の受取回数}$$

となる。つまり，年率 6% の預金で毎月 1 回利息を受け取れるとすれば，月率は $6\% \div 12 = 0.5\%$ となる。初期預金残高を 100 万円とすれば，1 年後，12 回目の利息を受け取った後には $1.005^{12} \times 100 = 106.1677\cdots$ 万円になっている。年率 6% の預金で毎日 1 回利息を受け取る場合 1 年後には，$(1 + 6\% \div 365)^{365} \times 100 = 1.06183\cdots$ 万円になる。当初資産額を S_0，利子率の年率を r，受け取り回数を N とすれば 1 年後の資産残高は

$$\left(1 + \frac{r}{N}\right)^N S_0$$

となる。S_0 にかかる式を次のように変形する。

$$\left(1 + \frac{r}{N}\right)^N = \left(1 + \frac{1}{N/r}\right)^N = \left[\left(1 + \frac{1}{N/r}\right)^{N/r}\right]^r$$

最右辺で r 乗される数が (1.2) と同じ形であることに注意してほしい：

$$n \longleftrightarrow \frac{N}{r}$$
$$\left(1 + \frac{1}{n}\right)^n \longleftrightarrow \left(1 + \frac{1}{N/r}\right)^{N/r}$$

$N \to \infty \Leftrightarrow N/r \to \infty$ だから，

$$\lim_{N\to\infty} \left(1 + \frac{r}{N}\right)^N = \lim_{N/r\to\infty} \left[\left(1 + \frac{1}{N/r}\right)^{N/r}\right]^r = e^r$$

となる。$N \to \infty$ というのは，すべての瞬間瞬間で利息を受け取れる仮想的な状況を表している。利息受け取りが時間的に連続した状況を考えているので，**連続複利計算**と呼ぶ。1 年後の資産残高は無限に大きくなる訳ではなく e^r 倍になることに注意しよう。なお，この 1 年間の変化

$$S_0 \longrightarrow S_0 e^r$$

について，対数差分を用いて変化率を求めると，

$$\begin{aligned}\log(S_0 e^r) - \log S_0 &= \log S_0 + \log e^r - \log S_0 \\ &= \log e^r \\ &= r\end{aligned}$$

となって利子率と一致する。瞬時的に受け取る利子率が（年率換算で）r であるとき（この利子率を「瞬時変化率」と呼んでおこう），1 年毎に資産は e^r 倍になる。対数差分は瞬時変化率 r に正確に一致する。

時間軸が飛び飛びの値を取る離散時間モデルに対して，時間軸が連続した値（つまり，実数値）を取るようモデル化の方法もある。そのようなモデルを「**連続時間モデル**」という。本書は主に離散時間モデルを扱うが，連続時間モデルの方が数学的な扱いが容易になることも多いため，離散時間・連続時間が状況に応じて使い分けられている。例えば，とある離散時間データの y_t から y_{t+1} への変化率を

$$\log y_{t+1} - \log y_t$$

で近似するのは，あたかも連続の時間軸上で成長したものと近似的に解釈して，瞬時変化率を計算していることになる。なお，瞬時変化率 r，あるいは同じことだが，離散時間の変化率 e^r が十分小さいときには，連続時間モデルと離散時間モデルの違いは無視できるほどに小さくなる。なぜなら，

$$e^r = \sum_{n=0}^{\infty} \frac{r^n}{n!} = 1 + r + \frac{r^2}{2!} + \frac{r^3}{3!} + \cdots \approx 1 + r$$

が成り立つからである。

瞬時変化率を使うのが便利なのは，瞬時変化率に次のような性質があるからだ。

事実 1.1. 変数 x と y はそれぞれ瞬時的に g_x, g_y で変化している。このとき，

1. 積 xy の瞬時変化率は $g_x + g_y$，
2. 商 x/y の瞬時変化率は $g_x - g_y$，

3. 任意の実数 α に対して，べき x^α の瞬時変化率は αg_x 。

離散時間モデルではこのようなきれいな性質は成り立たない。例えば，$x_{t+1} = (1+r_x)x_t, y_{t+1} = (1+r_y)y_t$ とすると，

$$\frac{x_{t+1}y_{t+1} - x_t y_t}{x_t y_t} = r_x + r_y + r_x r_y$$

となり，余分な項 $r_x r_y$ が現れる。この余分な項は r_x, r_y がともに十分小さければ，無視できるほどに小さくなる。

注意 1.7. 離散時間的にしか観測されない経済現象を分析するにあたっては，次のような近似的なアプローチがよく使われる。

- 離散時間的に観測される現象を連続時間的に観測されるものと見做して連続時間モデルを構築する。数学的に使いやすいモデルが得られる。
- 離散時間モデルを構築した上で，成長率に関して連続時間モデルに類似した公式が成り立つものと近似して，数学的な議論を簡略化する。

□

1.3 プログラミング

この章では Python を関数電卓として使う方法を学ぶ。まだプログラミングと呼ぶほどのものでもないが，単一の式の計算が次章以降の基本になる。

Windows を使っている人はスタートメニューから Anaconda PowerShell Prompt を探して開こう。Mac や Linux を使っている人はターミナル（端末）のアプリを開く。

Anaconda PowerShell Prompt を開くと次のような表示がでる。Mac や Linux の場合，Anaconda のインストール方法によっては同じような見た目にならないかもしれない。

```
(base) C:\Users\Kenji>
```

「>」（あるいは % や $ やもっと他の記号かもしれない）のような記号で終わるテキ

ストは「プロンプト」と呼ばれ，入力待ち状態を意味している。次のようにコマンド（ipython）を入力しよう。

```
(base) C:\Users\Kenji> ipython
```

入力が終わったら「Enter」あるいは「Return」と書かれたキーを押す。次のような表示が見えれば成功だ。

```
Python 3.7.5 (default, Nov  1 2019, 02:16:23)  Type 'copyright',
'credits' or 'license' for more information IPython 7.11.0 --
An enhanced Interactive Python. Type '?' for help.
In [1]:
```

IPython というインタラクティブに Python を使うためのアプリケーションが立ち上がった。IPython の入力待ち状態（プロンプト）は `In [n]:` という形式で表されている。`In [n]:` の後にコードを入力して，「Enter」あるいは「Return」と書かれたキーを押せばコードが実行され，表示すべき結果があれば即座に表示される。

以下，本書では IPython のプロンプトを省略する。

1.3.1　基本の計算

加算 +, 減算-, 乗算*, 除算/ の記号は使ったことがあるだろう。これらは `n + m` のように，数を 2 つ指定して使うので**二項演算子**（binary operator）と呼ばれる。

```
10 + 10.5
```

```
  20.5
```

```
3 - 5
```

```
  -2
```

```
2 * 3
```

```
  6
```

```
6 / 4
```

```
1.5
```

```
2 ** 10
```

```
1024
```

商（整数商）や剰余はそれぞれ二項演算子`//`,`%`を使う。

```
100 // 3
```

```
33
```

```
100 % 3
```

```
1
```

数 2 つで挟む二項演算子に対して，**単項演算子**（unary operator）は 1 つの数の前に置く。代表的なものには，負数を作る`-`がある。

```
-3 * 2
```

```
-6
```

```
2.5 * -2
```

```
-5.0
```

2 つ目のコードは，括弧を付けることで読みやすくできる。

```
2.5 * (-2)
```

```
-5.0
```

1.3.2 優先順位と丸括弧

1 つの式の中に複数の演算子が使われているときは優先順位の高い順に計算が実行される。優先順位が同じ演算子が並んでいるときは左から順番に実行される。優先順位の詳細は，Python 公式ドキュメント「演算子の優先順位[9]」の項を見てほしい。計算順序を変えたいときは丸括弧`(` `)`で囲む（波括弧や角括弧は，この目的では使えない）。ほとんどは数学の慣習と同じなので，特に迷うこともないだろうと思う。例えば，次のよ

[9] https://docs.python.org/ja/3/reference/expressions.html#operator-precedence

うな計算はすんなり理解できるだろう。

```
3 + 1 * 2
```

```
  5
```

```
(3 + 1) * 2
```

```
  8
```

```
6 / 2 / 3
```

```
  1.0
```

べき乗**，負符号-，積*，和 + はこの順序で優先される。つまり，べきは負符号より先に計算され，積は和よりも先に計算される。したがって，次の 2 つのコードは同じ意味である。

```
1 + - 1 * - 3 ** 2
```

```
  10
```

```
1 + (- 1) * (- (3 ** 2))
```

```
  10
```

前者よりも後者のコードの方が読みやすいと感じる人が多いだろうと思う。優先順位表を知らなくても計算できるからだ。多少ムダであっても，適切に括弧を付けて人間が読みやすいコードを書くことを心がけてほしい。

なお，丸括弧にはもう 1 つ，長い計算の途中で改行を入れて可読性を高める，という使い方があることを覚えておいてほしい。次のような使用法である。

```
(1 + 2 + 3 + 4 + 5
 + 6 + 7 + 8 + 9 + 10)
```

```
  55
```

同じ目的を達成するために括弧を使わず，バックスラッシュ\（日本語の環境では円マーク¥）を使うこともできる。

```
1 + 2 + 3 + 4 + 5\
  + 6 + 7 + 8 + 9 + 10
```

```
  55
```

1.3.3 変数

長い計算を実行するときには，重要な数字や計算途中の結果で意味があるものには名前を付けておくのが便利である。名前によって指し示そうとする対象（ここでは数字）をオブジェクト（object）と呼ぶ。オブジェクトに付けようとする名前のことを慣例的に**オブジェクト名**とか**変数**（variable）と呼ぶ[10]。Python リファレンスにおける正式名称は**識別子**（identifier）あるいは**名前**（name）である。しばしば「変数に値を代入する」と表現される操作には= という二項演算子を使う。**右辺のオブジェクトに左辺の名前を付ける**，という操作である。

下の 2 行のコードが実行していることは，次の 2 つのことである。

1.「10」という数字（整数）に `x` という名前をつける。
2. `x` という名前が指し示すものを参照する。

```
x = 10
x
```

```
10
```

今，私たちは IPython のコンソールで作業をしているので，`x` の中身である 10 が表示された。Python を実行している環境によっては，明示的に `print()` 関数を呼び出さないといけないかもしれない。

```
print(x)
```

```
10
```

変数は計算に使用することができる。

```
x * 10
```

```
100
```

[10] ここに書いている定義はかなりいい加減なので，真に受けないように。

成長の計算例

分析開始時点の GDP が 500 で，毎年 2% ずつ成長する経済を考えよう。3 年後の GDP を計算したい。

$$g = 0.02, \quad y_0 = 500, \quad t = 3$$

として，

$$y_t = (1 + g)^t y_0$$

である。素直に，数式をコードに置き換えて次のように書けばよい。

```
g = 0.02
y0 = 500
t = 3
yt = (1 + g) ** t * y0
yt
```

```
530.604
```

数式の表示ではわかりにくいという場合には，数字の意味を意識して次のようにしてもよい。

```
growth_rate = 0.02
GDP_0 = 500
years = 3
GDP_3 = (1 + growth_rate) ** years * GDP_0
GDP_3
```

```
530.604
```

どちらのコードがよりよいか，というのは一概には言えないのだが，数学的なモデルを数値的に分析するような状況では，数式との対応関係がはっきりと分かる前者のコードの方が保守がしやすい。一方，より汎用的なコードの開発という状況では，意味に応じた名前を付ける方が望ましいというケースもあるだろう。状況に応じた命名を心がけてほしい。

Python のオブジェクト名には次のルールがある。

- 大文字と小文字は区別する。

- 最初の文字として数字を使えない。
- 使える記号はアンダースコア_ だけ。
- Python のキーワードは変数名として使えない[*11]。

問題 1.7. ネイピア数の定義

$$e = \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n$$

に基づいて，近似的に e を計算してみよう。n を $n = 100, 1000, 10000$ と大きくしていったとき，$\left(1+\frac{1}{n}\right)^n$ はどのような数字に近づいていくか。

Answer

1.3.4 エラー

Python はエラーが発生するとコードの実行を停止し，

〇〇 Error: エラーの理由

のような形式でユーザーにエラーに関する情報提供をするようになっている。「〇〇 Error」はエラーの種類ごとに付けられた名前である。例えば，未定義の名前を参照しようとすると，\code{NameError}

が発生する。

```
print(abcde)
```

```
------------------------------------------------------------
NameError
```

[*11] True ,False,import,from,as,if,else,for,in など。全キーワードは Python の公式ドキュメントを参照のこと。https://docs.python.org/ja/3/reference/lexical_analysis.html#keywords

```
Traceback (most recent call last)
<ipython-input-17-e8409d1cf4b6> in <module>
----> 1 print(abcde)
NameError: name 'abcde' is not defined
```

「NameError: name 'abcde' is not defined」というところを読めば，abcde というありもしない名前にアクセスしようとしていることが原因と分かる。このように，エラーメッセージをきちんと読めば自力で問題解決できることが多い。

あなたが Python の初心者であれば，まずは次の 2 つのエラーに慣れてしまおう。

- NameError: 存在しない名前を参照するエラー
  - 名前は定義されているか？[*12]
  - スペルミスはないか？
  - 大文字・小文字の区別はできているか？
- SyntaxError: 書いたコードが Python の文法に則っていないというエラー
  - 名前のルールは守っているか？
  - キーワードを名前として使っていないか？
  - 開く括弧と閉じる括弧は正しく対応しているか？
  - その他，文法ルールは適切に守られているか？

これから多くのエラーに出会うことになるので，エラーメッセージを読む習慣を身につけよう。

問題 1.8. 次の 1〜5 のコードを実行したときに表示されるエラーメッセージに目を通し，「エラー名：エラーの理由」を書き取りなさい。エラーの原因について，自分の言葉で説明しなさい。

1. `3x = 10`
2. `True = 0`

[*12] 名前の定義の問題は Jupyter Notebook（後で紹介します）のような環境を使用する初心者を当惑させる原因になる。Jupyter Notebook を再起動して，途中からコードの実行を再開しようとするとこのエラーが出るのだ。確かに定義するコードはそこにあるのだけど，Python は知らないと言ってくる。再起動前に動いていたのであれば，コードを上から順番に実行し直させば解決するはずだ。

3. `a-b = 0`
4. `print(abcde)`
5. `{3 * (2 + 4)} * 3`

Answer

1.3.5 関数

脱線：関数とメソッド

複数の操作をまとめたり複雑な計算をするために「関数」を使うことができる。ユーザー（あなた）が関数を作ることもできる。複雑な計算を意味のある単位に分割して名前を付けておくことで，コードをクリーンに保つことができる。関数に定義された処理を実行することを「関数を呼び出す」とがある。2 つの関数呼び出しの方法があるので，一応ここで言及しておく。

- オブジェクトの外側から関数を作用させる方法。次の形式で呼び出す（`obj` はゼロ個でも，2 個以上でもよい。）

  ```
  function_name(obj)
  ```

- オブジェクトの内側から関数を呼び出す方法。ドットを使った次の形式で呼び出す。（other_obj はゼロ個でも，2 個以上でもよい。）

```
obj.function_name(other_obj)
```

「オブジェクトとは何か」というような質問に対する正確な回答を本書には期待しないでほしいのだけれど，1 つにはコンピュータおおざっぱに言えば

「オブジェクト」=「データ」+「データに定義された動作」

というイメージを持っておけばよい。

オブジェクトが持つ「動作」はもちろん関数の一種であるが，これを特に，**メソッド**と呼ぶことが多い[*13]。次のような例がある（何をやっているかはメソッドの名前を読めば分かる）。

```
1.5.as_integer_ratio()
```

```
(3, 2)
```

NumPy の関数

これ以上の例は今後のお楽しみということにしよう。Python は言語のコアの部分が非常に小さく，様々な興味のある処理を実行するためにライブラリと呼ばれる拡張機能を呼び出すことになる。ここで使うのは，**NumPy** という数値計算用のライブラリが持っている関数だ。Python の標準ライブラリ（必ずインストールされているライブラリ）には含まれていないが，数値計算を行うときの事実上の標準になっている[*14]。Anaconda を利用しているなら，**NumPy** はインストールされているはずだ。

ライブラリの拡張機能を呼び出すには，import 文を書く。いくつかの書き方があるが，多くの人は **NumPy** を次のようにロードする。

[*13] なお，ドットが付けば必ずメソッドかというとそうでもない。オブジェクトの中にさらに変数（**属性**という）が格納されている場合もある。説明が必要になったときに紹介しよう。

[*14] Python をインストールすると必ずインストールされる標準ライブラリにも **math** や **statistics** という数値計算目的のライブラリがある。**NumPy** を使えない環境で仕事をしなければならなくなったときに使い方を調べればよい。

```
import numpy as np
```

このコードを実行しても見かけ上は何も起こらないが，**NumPy** の関数を np.function_name() の形式で呼び出すことができるようになる。例えば，自然対数は np.log()，指数関数は np.exp() である。

```
np.log(10)
```

```
2.302585092994046
```

```
np.exp(1)
```

```
2.718281828459045
```

ここで **NumPy** の数学関数を網羅することはできないので，必要に応じて公式リファレンスを調べてほしい[*15]。種々の数学関数以外にも，重要な定数が定義されている。

```
np.pi
```

```
3.141592653589793
```

```
np.e
```

```
2.718281828459045
```

例 1.5. 成長率の近似公式（命題 1.3）を確認してみよう。ここでは，

$$y_0 = 300, \quad y_1 = 306$$

という数値例を使う。対数差分と通常の変化率の定義とが近い値を取ることを確認できる。

```
y0 = 300
y1 = 306
np.log(y1) - np.log(y0)
```

```
0.019802627296179764
```

```
(y1 - y0) / y0
```

[*15] https://docs.scipy.org/doc/numpy/reference/routines.math.html

```
0.02
```

問題 1.9. 他の数値例を用いて例 1.5 と同様のことを確かめなさい。y_0 から y_1 への変化率がどの程度の大きさであれば，命題 1.3 の近似公式は実用上使えそうか。

Answer