Отчет по лабораторной работе 2

Неполнодоступная двухсервисная модель Эрланга с одинаковыми интенсивностями обслуживания и зарезервированной емкостью

Генералов Даниил, 1032212280

0. теоретическая информация

Исследуется сота сети связи емкостью C. Пусть пользователям сети предоставляются услуги двух типов. Запросы в виде двух пуассоновский потоков (ПП) с интенсивностями λ_1, λ_2 поступают в соту. Среднее время обслуживания запросов на предоставление услуг каждого типа μ_1^{-1}, μ_2^{-1} соответственно. Исследуются основные характеристики модели для случая $\mu_1 = \mu_2 = \mu$.

Часть пропускной способности соты зарезервирована для обслуживания запросов на предоставление услуги 1-го или 2-го типа. Оставшаяся часть пропускной способности является полнодоступной для запросов на предоставление услуг обоих типов. Предположим, что сначала заполняется полнодоступная емкость.

В классификации Башарина-Кендалла: $M(\lambda_1)M(\lambda_2)|M(\mu_1)M(\mu_2)|C, g|0$.

- C -- пиковая пропускная способность соты;
- g -- полнодоступная часть пропускной способности соты;
- $C - g$ -- пропускная способность, зарезервированная для обслуживание запросов определенного типа;
- λ_1, λ_2 -- интенсивность поступления запросов на предоставление услуги 1, 2-го типа;
- μ^{-1} -- среднее время обслуживания запроса на предоставление услуги 1, 2-го типа;
- ρ_1, ρ_2 -- интенсивность предложенной нагрузки, создаваемой запросами на предоставление услуги 1, 2-го типа;
- $X(t)$ -- число запросов, обслуживаемых в системе в момент времени t, $t \geq 0$ (случайный процесс (СП), описывающий функционирование системы в момент времени t, $t \geq 0$);
- X -- пространство состояний системы;
- n -- число обслуживаемых в системе запросов;

- B_1, B_2 -- множество блокировок запросов на предоставление услуги 1, 2-го типа;
- S_1, S_2 -- множество приема запросов на предоставление услуги 1, 2-го типа.

Пусть часть пропускной способности соты $\mathrm{C} - g$ зарезервирована для обслуживания запросов на предоставление услуги 1-го типа.

Пространство состояний системы: $X = \{0, \ldots, C\}$; $|X| = C + 1$

Множество блокировок запросов на предоставление услуги i-типа, $i = 1, 2$: $B_1\{C\}$; $B_2 = \{g, g+1, \ldots, C\}$

Множество приема запросов на предоставление услуги i-типа, $i = 1, 2$: $S_i = \overline{B_i} = X \setminus B_i$; $S_1 = \{0, \ldots, C-1\}$; $S_2 = \{0, \ldots, C_2 - 1\}$

СУГБ:

- $(\lambda_1 + \lambda_2)p_0 = \mu p_1$
- $(\lambda_1 + \lambda_2 + n\mu)p_n = (\lambda_1 + \lambda_2)p_{n-1} + (n+1)\mu p_{n+1}$, $n = \overline{1, g-1}$
- $(\lambda_1 + g\mu)p_g = (\lambda_1 + \lambda_2)p_{g-1} + (g+1)\mu p_{g+1}$
- $(\lambda_1 + n\mu)p_n = \lambda_1 p_{n-1} + (n+1)\mu p_{n+1}$, $n = \overline{g+1, C-1}$
- $C\mu p_C = \lambda_1 p_{C-1}$

СУЛБ:

- $(\lambda_1 + \lambda_2)p_1 = n\mu p_n$, $n = \overline{1, g}$
- $\lambda_1 p_{n-1} = n\mu p_n$, $n = \overline{g+1, C}$

Обозначим $\rho_1 = \frac{\lambda_1}{\mu}$, $\rho_2 = \frac{\lambda_2}{\mu}$.

Стационарное распределение вероятностей состояний системы:

$p_n =$

- $p_0 * \frac{(\rho_1+\rho_2)^n}{n!}$, если $n = \overline{1, g}$;
- $p_0 * \frac{(\rho_1+\rho_2)^g * (\rho_1)^{n-g}}{n!}$, если $n = \overline{g+1, C}$.

При этом $p_0 = (\sum_{n=1}^{g} \frac{(\rho_1+\rho_2)^n}{n!} + \sum_{n=g+1}^{C} \frac{(\rho_1+\rho_2)^g * (\rho_1)^{n-g}}{n!})^{-1}$

Основные вероятностные характеристики модели:

- Вероятность блокировки по времени запроса на предоставление услуги 1-го типа: $E_1 = \sum_{n \in B_1} p_n = p_C$
- Вероятность блокировки по времени запроса на предоставление услуги 1-го типа: $E_2 = \sum_{n \in B_2} p_n = p_g + p_{g+1} + \ldots + p_C = \sum_{n=g}^{C} p_n$

- Среднее число обслуживаемых в системе запросов: $\overline{N} = \sum_{n \in X} np_n$

1. подключение библиотек, определение функций

Для расчета больших факториалов нам потребуется длинная арифметика, а для рисования графиков -- библиотека для визуализации данных.

In [2]:
```
:dep num = { version = "^0.4.3" }
:dep plotters = { version = "^0.3.6", default-features = false, features = [

extern crate num;
use num::BigRational as R;
use num::BigInt as I;
use num::BigUint as U;
use num::Integer;
use num::traits::ConstZero;
use num::FromPrimitive;
use num::ToPrimitive;

extern crate plotters;
use plotters::prelude::*;
```

Для удобства конвертации стандартных чисел в числа длинной арифметики используются helper-функции.

In [3]:
```
fn u(i: usize) -> U {
    U::from_usize(i).unwrap()
}

fn rr(i: f64) -> R {
    R::from_float(i).unwrap()
}
```

Для вычисления факториала нет стандартной функции, и очевидные подходы не работают с длинной арифметикой, поэтому эта функция считает это за нас.

Мы также проверяем, работает ли одно из правил факториалов: что $N! = (N-i)! + \prod_{x=N-i+1}^{N} x$.

In [4]:
```
use num::{BigInt, BigUint};
use num::ToPrimitive;
fn factorial(n: u8) -> BigUint {
    (1..=n).map(BigUint::from).product()
}

for i in 3..8 {
    assert_eq!(factorial(10), factorial(i) * (i+1 ..= 10).map(|x| x as usize
}
```

Out[4]: ()

Эта функция отображает график функции, принимая на вход список X-Y пар.

In [5]:
```
fn draw_chart(data: &Vec<(f32, f32)>, name: impl ToString) -> plotters::evcx
    let minx = data.iter().min_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(std
    let maxx = data.iter().max_by(|a, b| a.0.partial_cmp(&b.0).unwrap_or(std
    let miny = data.iter().min_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(std
    let maxy = data.iter().max_by(|a, b| a.1.partial_cmp(&b.1).unwrap_or(std
    let figure = evcxr_figure((640, 480), |root| {
        root.fill(&WHITE)?;
        let mut chart = ChartBuilder::on(&root)
            .caption(name.to_string(), ("Arial", 50).into_font())
            .margin(5)
            .x_label_area_size(30)
            .y_label_area_size(30)
            .build_cartesian_2d(minx..maxx, miny..maxy)?;

        chart.configure_mesh().draw()?;

        chart.draw_series(LineSeries::new(
            data.clone(),
            &RED,
        )).unwrap();

        // chart.configure_series_labels()
        //     .background_style(&WHITE.mix(0.8))
        //     .border_style(&BLACK)
        //     .draw()?;
        Ok(())
    });
    return figure;
}
```

Эта функция считает стационарное распределение вероятностей для рассматриваемой модели. Она принимает ρ_1 и ρ_2, которые соответствуют интенсивностям поступления заявок на два типа услуг, а также C -- максимальную пропускную способность, а именно количество заявок, которые могут одновременно выполняться, и g -- зарезервированная емкость для заявок первого типа.

Подсчет разделен на два случая: `prob_low` и `prob_high` -- первое относится к случаю, когда $i < g$, а второе к остальным случаям. Из-за этого `prob_low` используется в определении `prob_high`.

Функция возвращает список: i-й элемент списка равен вероятности, что система будет найдена в состоянии i (то есть это p_i).

In [6]:
```
/// Считает стационарное распределение вероятностей для пространства
fn stationary_prob_distribution(rho1: f64, rho2: f64, max_c: u8, g: u8) -> V
    let prob_low = |x| (rho1 + rho2).powf(x as f64) / (factorial(x).to_f64()
    let prob_high = |x| (
```

```
        prob_low(g) *
        rho1.powi(x as i32 - g as i32) /
        ((g+1) as usize..=x as usize).map(|x| x as f64).product::<f64>()
    );
    let prob = |x| if x<=g {prob_low(x)} else {prob_high(x)};

    let mut v = Vec::with_capacity(max_c as usize);
    let p0 = 1.0/(1..=max_c)
        // .map(|x| (println!("{x}"), x).1)
        .map(prob).sum::<f64>();

    for c in 0..=max_c {
        let res = p0 * prob(c);
        v.push(res);
    }
    v
}
```

2. входные параметры

Здесь задаются параметры, которые определяют модель. Чтобы попробовать запустить вычисления с другими значениями, вы можете поменять эту ячейку и перезапустить ее и все ячейки ниже.

При разработке я использовал следующие значения для теста:

- $\lambda 1 = 80$
- $\lambda 2 = 50$
- $\mu_1 = \mu_2 = 2$
- $C = 60$
- $g = 40$

In [7]:
```
let lambda1 = 80.0;
let lambda2 = 50.0;
let mu = 2.0;
let mu1 = mu;
let mu2 = mu;
let rho1 = lambda1 / mu1;
let rho2 = lambda2 / mu2;
let c = 60;
let g: usize = 40;
```

3. расчет

Распределение вероятностей можно посчитать с помощью нашей функции сверху. По свойству распределения вероятностей, сумма всех элементов должна быть равна 1, и мы здесь проверяем это.

In [9]:
```
let dist = stationary_prob_distribution(rho1, rho2, c as u8, g as u8);
println!("probability distribution: {dist:?}");
println!("Sum should be 1: {}", dist.iter().sum::<f64>());
```

```
probability distribution: [2.5542482810751626e-26, 1.6602613826988556e-24, 5.
395849493771281e-23, 1.1691007236504442e-21, 1.899788675931972e-20, 2.4697252
78711563e-19, 2.6755357186041938e-18, 2.4844260244181796e-17, 2.0185961448397
71e-16, 1.4578749934953903e-15, 9.476187457720037e-15, 5.599565315925476e-14,
3.0330978794596327e-13, 1.5165489397298164e-12, 7.041120077317004e-12, 3.0511
520335040355e-11, 1.2395305136110144e-10, 4.739381375571525e-10, 1.7114432745
119397e-9, 5.854937518067162e-9, 1.9028546933718278e-8, 5.889788336627086e-8,
1.7401647358216393e-7, 4.917856862104632e-7, 1.3319195668200045e-6, 3.4629908
737320113e-6, 8.657477184330028e-6, 2.084207470301674e-5, 4.838338770343171e-
5, 0.00010844552416286419, 0.00023496530235287238, 0.000492669182352797, 0.00
1000734276654119, 0.001971143272197507, 0.0037683621380246446, 0.006998386827
7600555, 0.01263597621678899, 0.022198336597061736, 0.03797083891602666, 0.06
32847315267111, 0.10283768873090554, 0.10032945242039565, 0.0955518594479958
4, 0.08888545064929847, 0.08080495513572587, 0.07182662678731189, 0.062457936
336792946, 0.05315569049939826, 0.04429640874949854, 0.03616033367306004, 0.0
28928266938448032, 0.02268883681446904, 0.017452951395745417, 0.0131720387892
41823, 0.009757065769808759, 0.007096047832588188, 0.005068605594705848, 0.00
35569162068111214, 0.002453045659869739, 0.001663081803301518, 0.001108721202
201012]
Sum should be 1: 1
```

Среднее число запросов, которые находятся в обработке -- это математическое ожидание набора вероятностей: ((вероятность того, что там 1 заявка) * 1 + (вероятность того, что там 2 заявки) * 2 + ...) / (макс. заявок)

Чтобы посчитать это, мы считаем сумму значений вероятности, помноженных на свой индекс (который равен количеству заявок, соответствующих этой ячейке).

In [10]:
```
let avg: f64 = dist.iter().enumerate().map(|(i,v)| (i as f64) * v).sum();
println!("Average requests in flight: {avg}");
```

```
Average requests in flight: 43.72435164097261
```

Вероятность блокировки по времени -- это вероятность того, что система оказывается в состоянии, когда нельзя обработать ни одну заявку, пока нагрузка не спадет.

Для заявок первого типа это -- только случай, когда нет больше ячеек вообще (то есть p_C), а для второго типа это все случаи, когда обрабатывается больше чем g заявок (потому что та пропускная способность зарезервирована для заявок первого типа).

In [11]:
```
let time_blocking_prob1 = dist.last().copied().unwrap();
let time_blocking_prob2 = dist.iter().skip(g).sum::<f64>();
println!("time blocking probability (E1): {time_blocking_prob1}");
println!("time blocking probability (E2): {time_blocking_prob2}");
```

```
time blocking probability (E1): 0.001108721202201012
time blocking probability (E2): 0.8492519804375737
```

Вероятность блокировки по вызовам -- это вероятность, что определенный из двух типов запросов будет заблокирован; поэтому это связано с вероятностью поступления каждого из двух запросов.

In [12]:
```
let req_blocking_prob1 = lambda1 / (lambda1 + lambda2) * time_blocking_prob1
let req_blocking_prob2 = lambda2 / (lambda1 + lambda2) * time_blocking_prob1
println!("request blocking probability: {req_blocking_prob1}, {req_blocking_
```

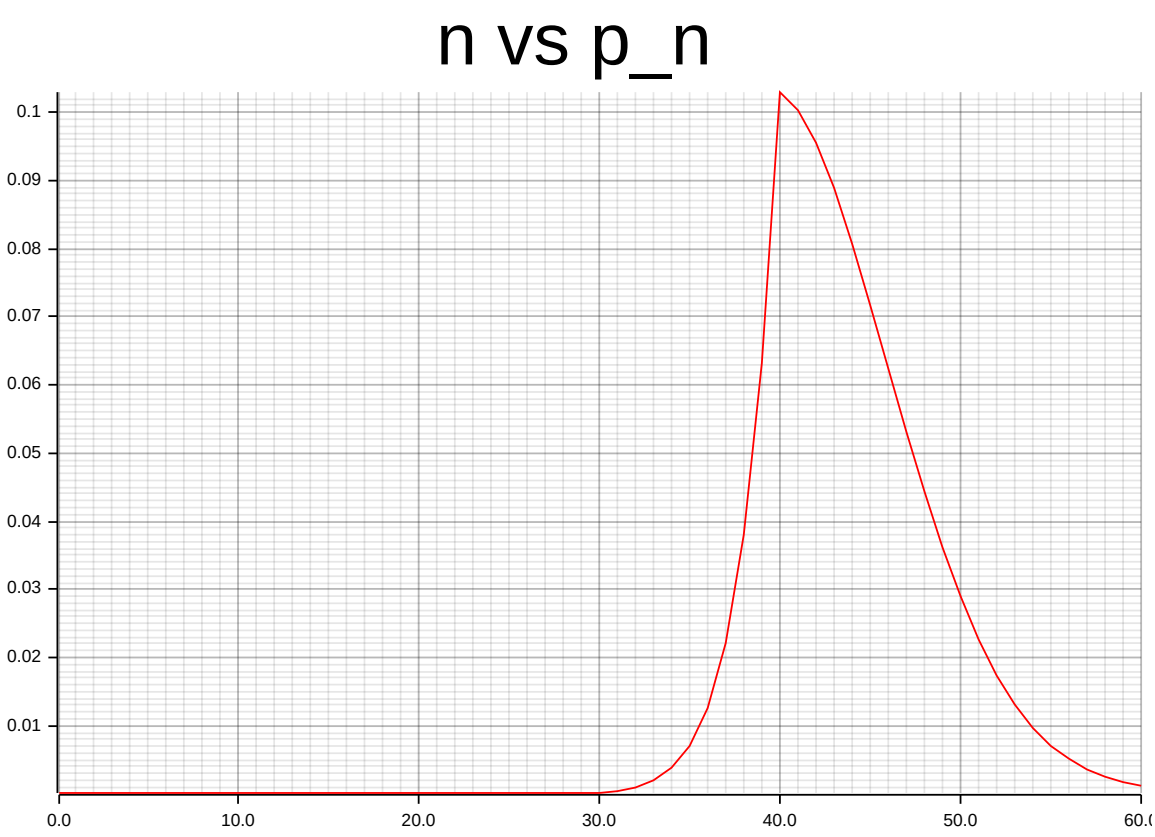
```
request blocking probability: 0.0006822899705852383, 0.0004264312316157739
```

4. графики

График распределения вероятности показывает, какие состояния системы являются самыми частыми. Если пик этого графика находится на правой границе, значит система перегружена; иначе он показывает среднее количество ожидающих своей очереди запросов (таким образом это значение имеет похожий смысл на E).

In [13]:
```
draw_chart(&dist.iter().enumerate().map(|(x,y)| (x as f32,*y as f32)).collec
```

Out[13]:



Для того, чтобыв посчитать график зависимости вероятности блокировки от интенсивности, надо зафиксировать значение одной из переменных λ и

варьировать другую; для каждой такой конфигурации нужно вычислить значение E и затем отобразить на графике.

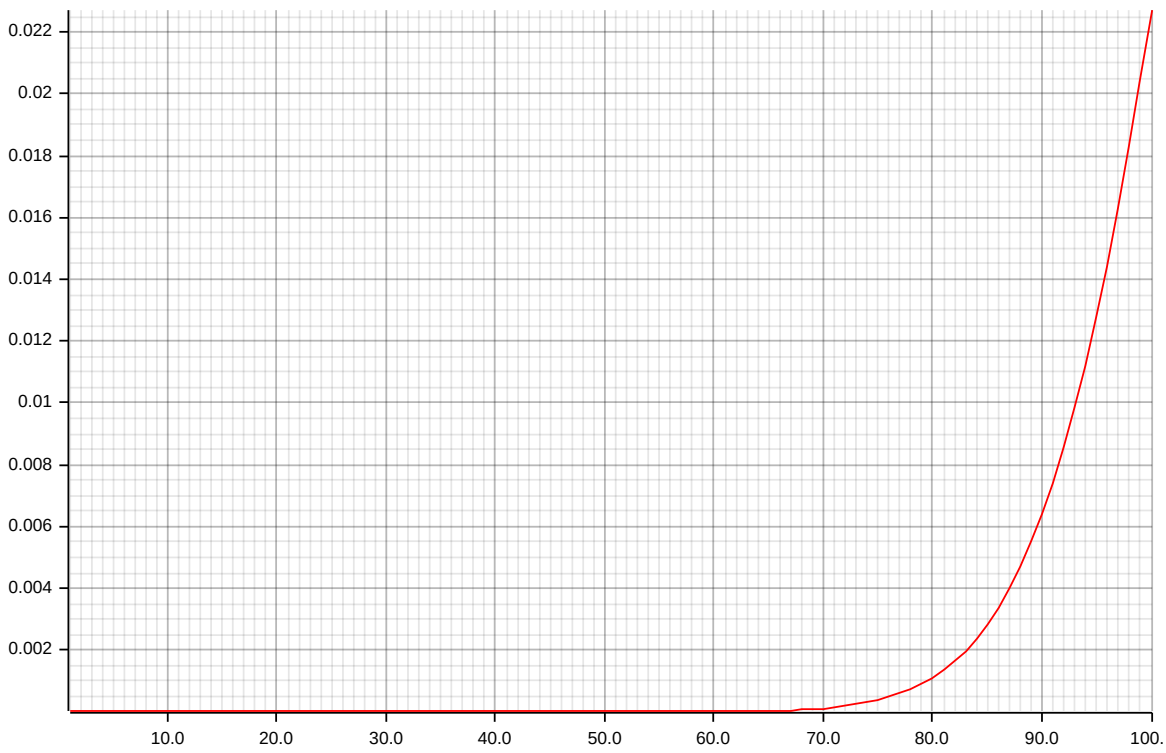
Сначала делаем это для λ_1.

In [15]:
```
let mut block_prob = vec![];
for lambda1 in 1..=100 {
    let rho1 = lambda1 as f64 / &mu1;
    let dist = stationary_prob_distribution(rho1, rho2, c as u8, g as u8);
    block_prob.push((lambda1 as f32, dist.last().cloned().unwrap() as f32));
}
block_prob
```

Out[15]:
```
[(1.0, 1.77e-43), (2.0, 2.4478834e-37), (3.0, 1.0591664e-33), (4.0, 4.28617
9e-31), (5.0, 4.7081145e-29), (6.0, 2.2570824e-27), (7.0, 6.0856095e-26),
(8.0, 1.0738625e-24), (9.0, 1.3677995e-23), (10.0, 1.3447435e-22), (11.0,
1.0705925e-21), (12.0, 7.151608e-21), (13.0, 4.1184912e-20), (14.0, 2.08837
33e-19), (15.0, 9.482566e-19), (16.0, 3.9085558e-18), (17.0, 1.4789452e-1
7), (18.0, 5.1855205e-17), (19.0, 1.6980775e-16), (20.0, 5.2282743e-16), (2
1.0, 1.5222797e-15), (22.0, 4.212393e-15), (23.0, 1.112621e-14), (24.0, 2.8
158101e-14), (25.0, 6.851038e-14), (26.0, 1.6073124e-13), (27.0, 3.6457583e
-13), (28.0, 8.014026e-13), (29.0, 1.7108645e-12), (30.0, 3.5540039e-12),
(31.0, 7.196365e-12), (32.0, 1.4226139e-11), (33.0, 2.7495598e-11), (34.0,
5.20248e-11), (35.0, 9.64826e-11), (36.0, 1.7557211e-10), (37.0, 3.1381067e
-10), (38.0, 5.5142735e-10), (39.0, 9.534293e-10), (40.0, 1.6233456e-9), (4
1.0, 2.723784e-9), (42.0, 4.5067874e-9), (43.0, 7.3581092e-9), (44.0, 1.186
1042e-8), (45.0, 1.888738e-8), (46.0, 2.9725653e-8), (47.0, 4.6259967e-8),
(48.0, 7.12169e-8), (49.0, 1.0850298e-7), (50.0, 1.6366135e-7), (51.0, 2.44
48414e-7), (52.0, 3.618256e-7), (53.0, 5.306725e-7), (54.0, 7.7153976e-7),
(55.0, 1.1122747e-6), (56.0, 1.5903685e-6), (57.0, 2.2558906e-6), (58.0, 3.
175183e-6), (59.0, 4.4354697e-6), (60.0, 6.150551e-6), (61.0, 8.467788e-6),
(62.0, 1.1576572e-5), (63.0, 1.5718519e-5), (64.0, 2.1199596e-5), (65.0, 2.
8404434e-5), (66.0, 3.781301e-5), (67.0, 5.00199e-5), (68.0, 6.5756234e-5),
(69.0, 8.591444e-5), (70.0, 0.00011157569), (71.0, 0.00014404001), (72.0,
0.0001848587), (73.0, 0.0002358686), (74.0, 0.0002992276), (75.0, 0.0003774
505), (76.0, 0.0004734441), (77.0, 0.00059054035), (78.0, 0.0007325257), (7
9.0, 0.0009036656), (80.0, 0.0011087212), (81.0, 0.0013529578), (82.0, 0.00
16421421), (83.0, 0.0019825266), (84.0, 0.002380821), (85.0, 0.0028441472),
(86.0, 0.00337998), (87.0, 0.003996072), (88.0, 0.0047003627), (89.0, 0.005
500877), (90.0, 0.0064056087), (91.0, 0.007422402), (92.0, 0.008558822), (9
3.0, 0.009822028), (94.0, 0.011218651), (95.0, 0.012754676), (96.0, 0.01443
534), (97.0, 0.016265037), (98.0, 0.01824725), (99.0, 0.020384496), (100.0,
0.022678297)]
```

In [16]:
```
draw_chart(&block_prob, "lambda1 vs E")
```

Out[16]:



После этого -- для λ_2.

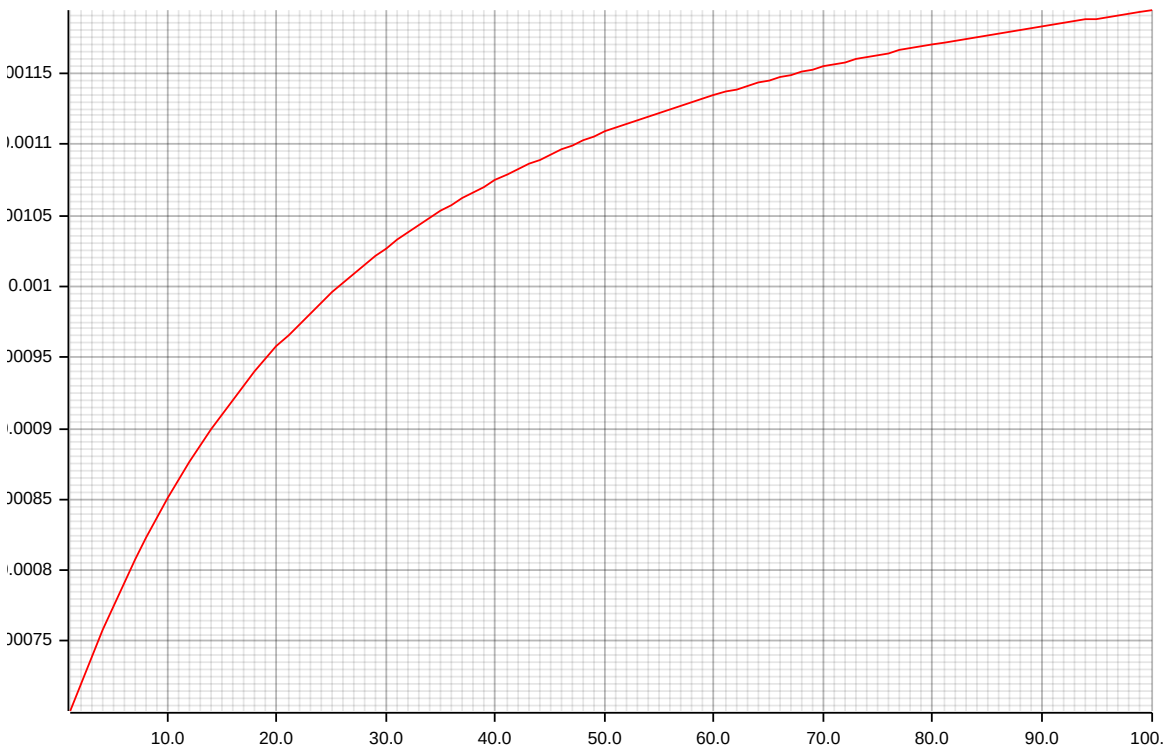
In [17]:

```
let mut block_prob = vec![];
for lambda2 in 1..=100 {
    let rho2 = lambda2 as f64 / &mu1;
    let dist = stationary_prob_distribution(rho1, rho2, c as u8, g as u8);
    block_prob.push((lambda2 as f32, dist.last().cloned().unwrap() as f32));
}
block_prob
```

Out[17]:
```
[(1.0, 0.0007004019), (2.0, 0.0007204047), (3.0, 0.0007394996), (4.0, 0.000
75771636), (5.0, 0.0007750877), (6.0, 0.00079164817), (7.0, 0.0008074333),
(8.0, 0.0008224789), (9.0, 0.0008368207), (10.0, 0.0008504938), (11.0, 0.00
086353236), (12.0, 0.0008759696), (13.0, 0.0008878374), (14.0, 0.000899166
3), (15.0, 0.00090998545), (16.0, 0.0009203226), (17.0, 0.00093020406), (1
8.0, 0.00093965477), (19.0, 0.00094869814), (20.0, 0.00095735653), (21.0,
0.0009656507), (22.0, 0.0009736004), (23.0, 0.0009812242), (24.0, 0.0009885
395), (25.0, 0.0009955628), (26.0, 0.0010023093), (27.0, 0.0010087935), (2
8.0, 0.001015029), (29.0, 0.0010210287), (30.0, 0.0010268046), (31.0, 0.001
0323678), (32.0, 0.0010377292), (33.0, 0.0010428985), (34.0, 0.0010478852),
(35.0, 0.0010526981), (36.0, 0.0010573457), (37.0, 0.0010618357), (38.0, 0.
0010661754), (39.0, 0.0010703718), (40.0, 0.0010744317), (41.0, 0.001078361
1), (42.0, 0.0010821657), (43.0, 0.0010858513), (44.0, 0.0010894231), (45.
0, 0.001092886), (46.0, 0.0010962446), (47.0, 0.0010995032), (48.0, 0.00110
26664), (49.0, 0.0011057378), (50.0, 0.0011087212), (51.0, 0.0011116203),
(52.0, 0.0011144385), (53.0, 0.0011171789), (54.0, 0.0011198446), (55.0, 0.
0011224386), (56.0, 0.0011249635), (57.0, 0.001127422), (58.0, 0.001129816
6), (59.0, 0.0011321497), (60.0, 0.0011344235), (61.0, 0.0011366402), (62.
0, 0.0011388018), (63.0, 0.0011409104), (64.0, 0.0011429678), (65.0, 0.0011
449758), (66.0, 0.001146936), (67.0, 0.0011488502), (68.0, 0.0011507199),
(69.0, 0.0011525466), (70.0, 0.0011543317), (71.0, 0.0011560766), (72.0, 0.
0011577826), (73.0, 0.001159451), (74.0, 0.0011610829), (75.0, 0.001162679
5), (76.0, 0.0011642419), (77.0, 0.0011657713), (78.0, 0.0011672686), (79.
0, 0.0011687347), (80.0, 0.0011701707), (81.0, 0.0011715774), (82.0, 0.0011
729557), (83.0, 0.0011743064), (84.0, 0.0011756304), (85.0, 0.0011769284),
(86.0, 0.0011782012), (87.0, 0.0011794494), (88.0, 0.0011806737), (89.0, 0.
0011818749), (90.0, 0.0011830536), (91.0, 0.0011842104), (92.0, 0.001185345
8), (93.0, 0.0011864605), (94.0, 0.001187555), (95.0, 0.0011886299), (96.0,
0.0011896857), (97.0, 0.0011907227), (98.0, 0.0011917417), (99.0, 0.0011927
43), (100.0, 0.001193727)]
```

In [18]:
```
draw_chart(&block_prob, "lambda2 vs E")
```

Out[18]:



Для того, чтобы нарисовать график среднего числа обслуживаемых запросов, нужно сделать то же самое: по разным значениям λ посчитать распределение вероятности, из него -- среднее количество, и отобразить для него точку. Сначала делаем это, варьируя λ_1, а затем λ_2.

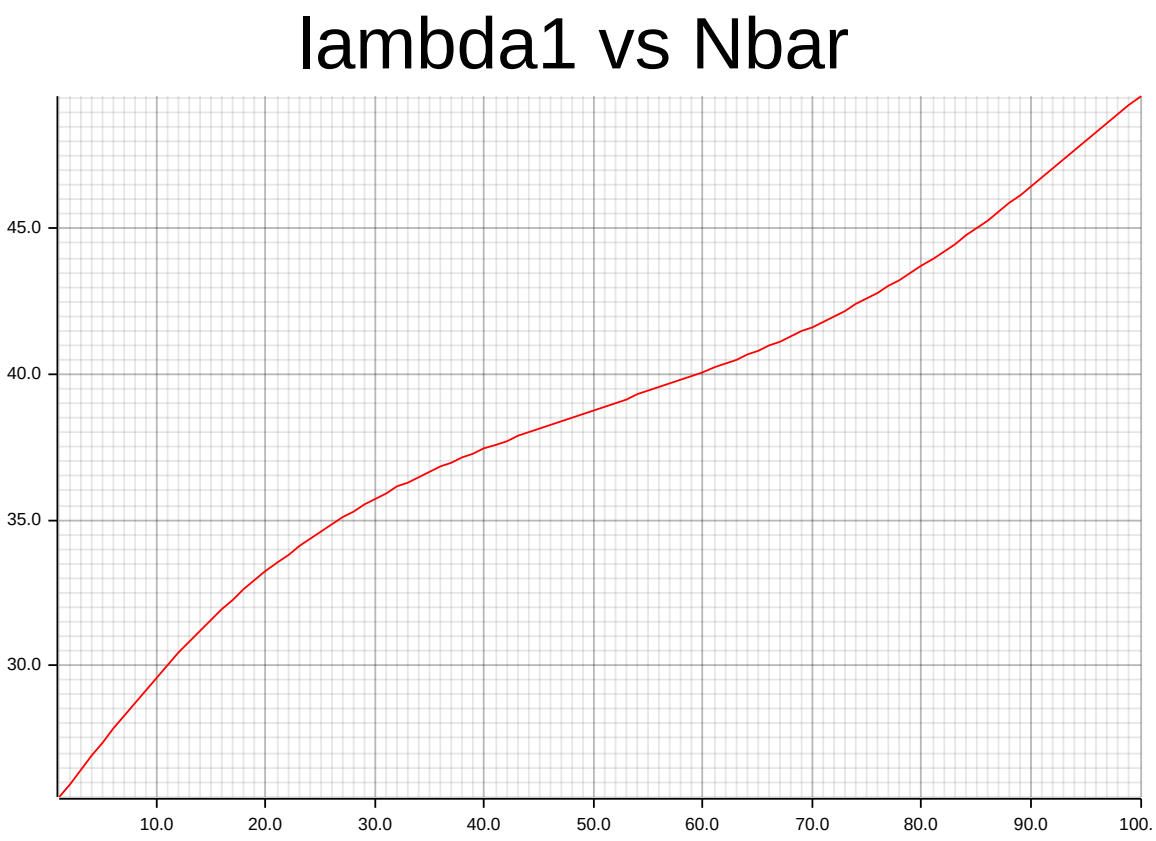
In [19]:

```
let mut avg_req_counts = vec![];
for lambda1 in 1..=100 {
    let rho1 = (lambda1 as f64) / &mu1;
    let dist = stationary_prob_distribution(rho1, rho2, c as u8, g as u8);
    let avg: f64 = dist.iter().enumerate().map(|(i,v)| i as f64 * v).sum();
    avg_req_counts.push((lambda1 as f32, avg as f32));
}
avg_req_counts
```

Out[19]:
```
[(1.0, 25.452143), (2.0, 25.93603), (3.0, 26.415709), (4.0, 26.890446), (5.
0, 27.35946), (6.0, 27.821955), (7.0, 28.277117), (8.0, 28.724144), (9.0, 2
9.162264), (10.0, 29.590746), (11.0, 30.00892), (12.0, 30.416199), (13.0, 3
0.812069), (14.0, 31.196117), (15.0, 31.568027), (16.0, 31.927584), (17.0,
32.274662), (18.0, 32.60924), (19.0, 32.93138), (20.0, 33.241222), (21.0, 3
3.538982), (22.0, 33.824936), (23.0, 34.09942), (24.0, 34.3628), (25.0, 34.
615498), (26.0, 34.85795), (27.0, 35.09061), (28.0, 35.313957), (29.0, 35.5
28473), (30.0, 35.734642), (31.0, 35.932945), (32.0, 36.123863), (33.0, 36.
307865), (34.0, 36.485416), (35.0, 36.656967), (36.0, 36.822956), (37.0, 3
6.983807), (38.0, 37.139935), (39.0, 37.291744), (40.0, 37.439613), (41.0,
37.583927), (42.0, 37.72504), (43.0, 37.863308), (44.0, 37.99907), (45.0, 3
8.13266), (46.0, 38.264397), (47.0, 38.394592), (48.0, 38.523563), (49.0, 3
8.6516), (50.0, 38.779007), (51.0, 38.90607), (52.0, 39.03308), (53.0, 39.1
60324), (54.0, 39.288082), (55.0, 39.41664), (56.0, 39.54628), (57.0, 39.67
7288), (58.0, 39.809944), (59.0, 39.94454), (60.0, 40.081352), (61.0, 40.22
068), (62.0, 40.36281), (63.0, 40.508038), (64.0, 40.65666), (65.0, 40.8089
68), (66.0, 40.965263), (67.0, 41.12584), (68.0, 41.290997), (69.0, 41.4610
18), (70.0, 41.636196), (71.0, 41.816803), (72.0, 42.00311), (73.0, 42.1953
7), (74.0, 42.393818), (75.0, 42.598667), (76.0, 42.810104), (77.0, 43.0282
9), (78.0, 43.253345), (79.0, 43.485355), (80.0, 43.72435), (81.0, 43.9703
2), (82.0, 44.2232), (83.0, 44.482857), (84.0, 44.749104), (85.0, 45.02168
7), (86.0, 45.300285), (87.0, 45.584522), (88.0, 45.87394), (89.0, 46.1680
3), (90.0, 46.466225), (91.0, 46.767902), (92.0, 47.072395), (93.0, 47.37
9), (94.0, 47.686993), (95.0, 47.995613), (96.0, 48.304104), (97.0, 48.6117
1), (98.0, 48.917686), (99.0, 49.22131), (100.0, 49.521885)]
```

In [20]:
```
draw_chart(&avg_req_counts, "lambda1 vs Nbar")
```

Out[20]:

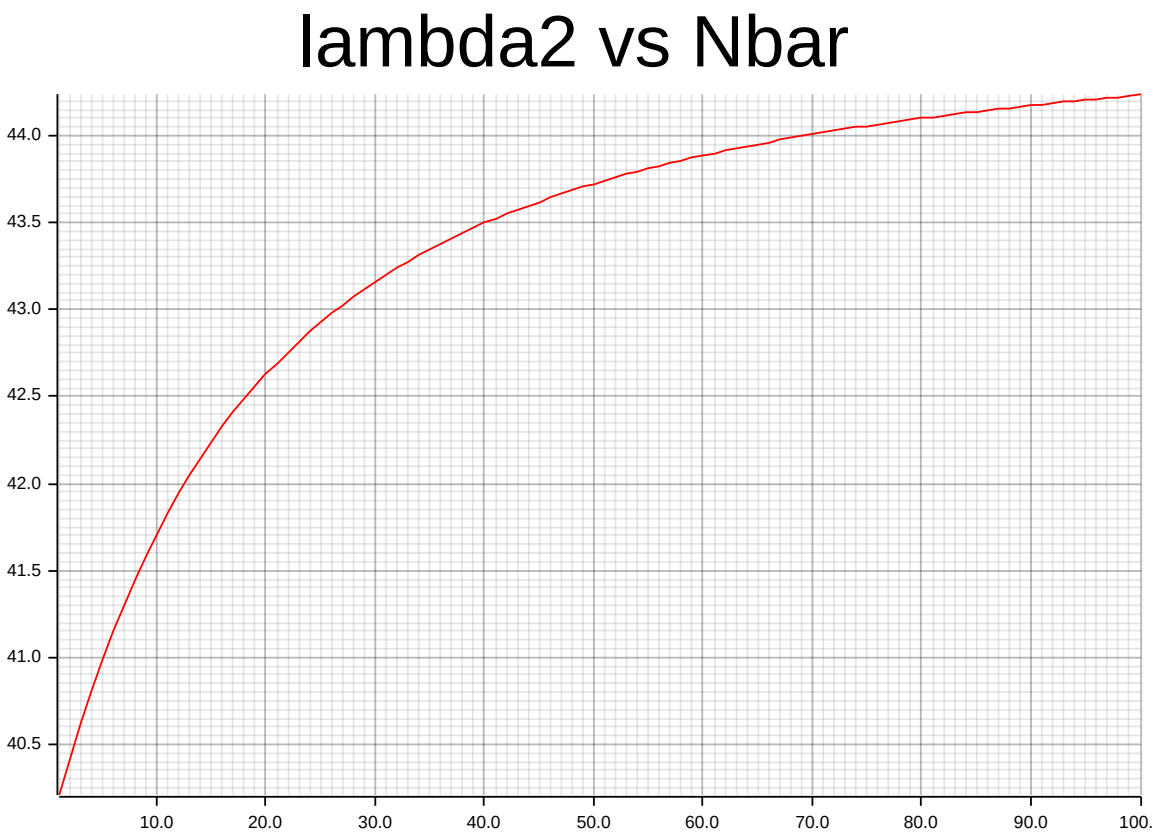


In [21]:
```rust
let mut avg_req_counts = vec![];
for lambda2 in 1..=100 {
    let rho2 = (lambda2 as f64) / &mu2;
    let dist = stationary_prob_distribution(rho1, rho2, c as u8, g as u8);
    let avg: f64 = dist.iter().enumerate().map(|(i,v)| i as f64 * v).sum();
    avg_req_counts.push((lambda2 as f32, avg as f32));
}
avg_req_counts
```

Out[21]:
```
[(1.0, 40.20374), (2.0, 40.419373), (3.0, 40.62076), (4.0, 40.80891), (5.0,
40.984753), (6.0, 41.14919), (7.0, 41.303047), (8.0, 41.44711), (9.0, 41.58
2104), (10.0, 41.7087), (11.0, 41.82752), (12.0, 41.93914), (13.0, 42.04409
4), (14.0, 42.142864), (15.0, 42.23591), (16.0, 42.32364), (17.0, 42.40643
7), (18.0, 42.48465), (19.0, 42.558605), (20.0, 42.6286), (21.0, 42.69490
4), (22.0, 42.757774), (23.0, 42.817436), (24.0, 42.874107), (25.0, 42.9279
82), (26.0, 42.979248), (27.0, 43.028065), (28.0, 43.074593), (29.0, 43.118
973), (30.0, 43.16134), (31.0, 43.201813), (32.0, 43.24051), (33.0, 43.2775
3), (34.0, 43.31298), (35.0, 43.346947), (36.0, 43.379513), (37.0, 43.4107
6), (38.0, 43.440758), (39.0, 43.469578), (40.0, 43.49728), (41.0, 43.5239
3), (42.0, 43.54958), (43.0, 43.57428), (44.0, 43.59808), (45.0, 43.62102
5), (46.0, 43.64316), (47.0, 43.664524), (48.0, 43.685154), (49.0, 43.70508
6), (50.0, 43.72435), (51.0, 43.74298), (52.0, 43.76101), (53.0, 43.77845
8), (54.0, 43.795357), (55.0, 43.811726), (56.0, 43.82759), (57.0, 43.8429
8), (58.0, 43.857903), (59.0, 43.872383), (60.0, 43.886444), (61.0, 43.900
1), (62.0, 43.913364), (63.0, 43.926258), (64.0, 43.938793), (65.0, 43.9509
85), (66.0, 43.962845), (67.0, 43.97439), (68.0, 43.985626), (69.0, 43.9965
7), (70.0, 44.007233), (71.0, 44.01762), (72.0, 44.027744), (73.0, 44.0376
2), (74.0, 44.04725), (75.0, 44.056644), (76.0, 44.06581), (77.0, 44.0747
6), (78.0, 44.083496), (79.0, 44.09203), (80.0, 44.100365), (81.0, 44.1085
1), (82.0, 44.11647), (83.0, 44.124252), (84.0, 44.131863), (85.0, 44.13930
5), (86.0, 44.146584), (87.0, 44.153706), (88.0, 44.16068), (89.0, 44.16750
3), (90.0, 44.174187), (91.0, 44.18073), (92.0, 44.18714), (93.0, 44.1934
2), (94.0, 44.199574), (95.0, 44.205605), (96.0, 44.211517), (97.0, 44.2173
1), (98.0, 44.222996), (99.0, 44.228573), (100.0, 44.23404)]
```

In [22]:
```rust
draw_chart(&avg_req_counts, "lambda2 vs Nbar")
```

Out[22]:


lambda2 vs Nbar
44.0
43.5
43.0
42.5
42.0
41.5
41.0
40.5
10.0
20.0
30.0
40.0
50.0
60.0
70.0
80.0
90.0
100.

In [ ]: