

Отчет по лабораторной работе 4

Неполнодоступная двухсервисная модель Эрланга с резервированием для заявок второго типа и разными интенсивностями обслуживания. Первыми заполняются зарезервированные приборы.

Генералов Даниил, 1032212280

Рассматривается неполнодоступная двухсервисная модель Эрланга с разными интенсивностями обслуживания и резервированием для обслуживания запросов на предоставление услуги 2-го типа. Запросы на предоставление услуги 2-го типа сначала заполняют зарезервированную емкость.

- C -- общее число приборов;
- g -- число полностью доступных приборов;
- $C - g$ -- число зарезервированных приборов;
- λ_1, λ_2 -- интенсивность поступления запросов 1, 2-го типа;
- μ^{-1} -- среднее время обслуживания запроса 1, 2-го типа;
- ρ_1, ρ_2 -- интенсивность предложенной нагрузки, создаваемой запросами 1, 2-го типа;
- $X(t)$ -- число запросов, обслуживаемых в системе в момент времени $t, t \geq 0$;
- X -- пространство состояний системы;
- n -- число обслуживаемых в системе запросов;
- B_1, B_2 -- множество блокировок запросов 1, 2-го типа;
- S_1, S_2 -- множество приема запросов 1, 2-го типа.

1. подключение библиотек, определение функций

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

```
In [3]: :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 [4]: 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.

```

In [5]: fn factorial(n: &U) -> R {
        let mut c = n.clone();
        let one = I::from_i8(1).unwrap();
        let mut out = R::new(one.clone(), one.clone());
        while c > U::ZERO {
            out *= R::new(I::from_biguint(num::bigint::Sign::Plus, c.clone()), o
            c -= 1u32;
        }
        out
    }

```

Эта функция возвращает список, состоящий из только уникальных значений в указанном списке. Это нужно для того, чтобы упростить вычисление набора состояний.

```

In [6]: fn make_unique<T: Eq + std::hash::Hash>(v: Vec<T>) -> Vec<T> {
        let mut set = std::collections::HashSet::new();
        for x in v {
            set.insert(x);
        }
        set.into_iter().collect()
    }

```

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

```

In [7]: 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;
}

```

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:

- $C = 4$
- $g = 3$
- $\rho_1 = \frac{1}{2}$
- $\rho_2 = 1$
- $\mu = 10$

```

In [8]: let C = 4;
let g = 3;
let rho_1 = 0.5;
let rho_2 = 1.0;

let mu1 = 10.0;

```

3. вычисления

Для того, чтобы определить набор состояний, мы рассматриваем все возможные пары $(x, y) \geq (0, 0)$, где $x + y \leq C$ и $n_1 \leq g$. Этот цикл находит их и сохраняет в список.

```
In [9]: println!("X = (n1, n2): n1 = 0,{g}, n2=0,{C}, n1+n2 <= {C}");
let mut states = vec![];
for n1 in 0..=g {
    for n2 in 0..=C {
        if n1 + n2 <= C {
            states.push((n1, n2));
        }
    }
}
println!("|X| = {}", states.len());
```

```
X = (n1, n2): n1 = 0,3, n2=0,4, n1+n2 <= 4
|X| = 14
```

Для того, чтобы показать набор состояний, мы должны нарисовать граф, который содержит все состояния в наборе X, и имеет соединения между соседними ячейками. Этот код формирует такой граф, затем визуализирует его через Graphviz, и затем показывает в виде картинки.

```
In [10]: :dep graphviz-rust = { version = "^0.9.0" }
:dep base64 = { version = "^0.22.1" }
use base64::prelude::*;

let mut dot_graph = String::new();

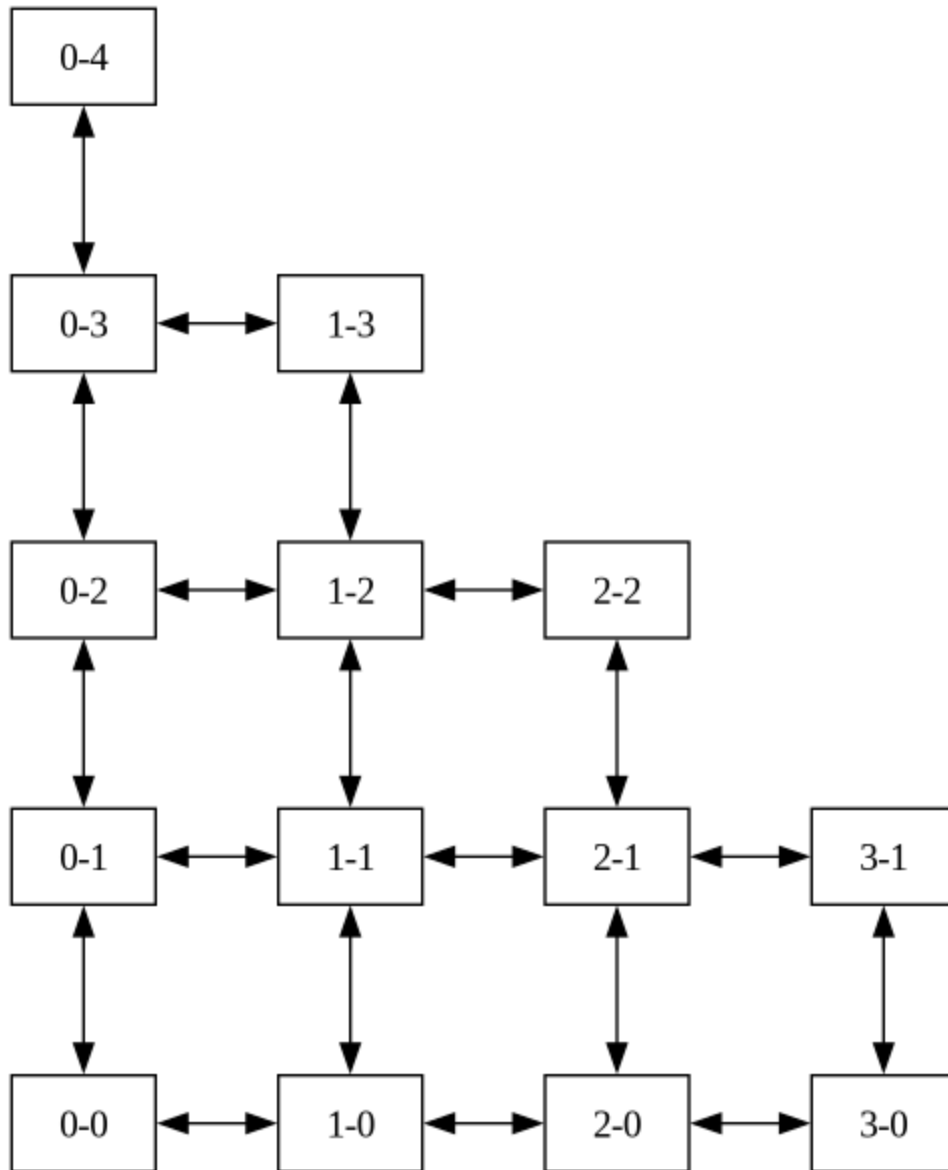
dot_graph.push_str("graph G {\n");
dot_graph.push_str("    rankdir=LR;\n");
dot_graph.push_str("    layout=nop;\n");
dot_graph.push_str("    node [shape=box];\n");

for (n1, n2) in states.iter() {
    dot_graph.push_str(&format!("S{}_{} [label=\"{}-{}\", pos=\"{:}.1\", {:.1}
let (n1, n2) = (*n1, *n2);
if states.contains(&(n1+1, n2)) {
    dot_graph.push_str(&format!("S{}_{} -- S{}_{} [dir=both]\n", n1, n2,
}
if states.contains(&(n1, n2+1)) {
    dot_graph.push_str(&format!("S{}_{} -- S{}_{} [dir=both]\n", n1, n2,
}
}

dot_graph.push_str("}\n");

let png = graphviz_rust::exec_dot(dot_graph, vec![graphviz_rust::cmd::Format
let png_base64 = BASE64_STANDARD.encode(&png);
println!("EVCXR_BEGIN_CONTENT image/png\n{}\nEVCXR_END_CONTENT", png_base64)
```

Out[10]:



Состояния блокировки заявок -- это те, находясь в которых, система не может принять больше заявок первого или второго типов. Для обоих типов заявок таковыми являются те, когда общее количество заявок равно C (тогда даже приоритетные заявки нельзя обработать), но для заявок первого типа -- есть еще ситуация, когда заняты все выделенные слоты (когда количество заявок $n_1 = g$).

```

In [11]: // boundary states
let mut b1 = vec![];
let mut b2 = vec![];
for (n1, n2) in states.iter().cloned() {
    if n1 == g {
        b1.push((n1, n2));
    }
    if n1 + n2 == C {
        b1.push((n1, n2));
        b2.push((n1, n2));
    }
}

```

```

let b1 = make_unique(b1);
let b2 = make_unique(b2);
println!("B1 = (n1, n2) \\\in X: n1 = {g} \\\lor n1 + n2 = {C} = {b1:?}");
println!("B2 = (n1, n2) \\\in X: n1 + n2 = {C} = {b2:?}");

```

B1 = (n1, n2) \\\in X: n1 = 3 \\\lor n1 + n2 = 4 = [(3, 1), (3, 0), (0, 4), (1, 3), (2, 2)]

B2 = (n1, n2) \\\in X: n1 + n2 = 4 = [(1, 3), (0, 4), (2, 2), (3, 1)]

Множество приема запросов -- это обратное множество к множеству блокировки:

$S_i = X \setminus B_i$. Здесь мы вычисляем его именно так, но также есть способ написать обратную формулу к формуле определения множества блокировки.

```

In [12]: // interior states
let mut s1 = vec![];
let mut s2 = vec![];
for s in states.iter() {
    if !b1.contains(s) {
        s1.push(*s);
    }
    if !b2.contains(s) {
        s2.push(*s);
    }
}

println!("S1 = (n1, n2) \\\in X: n1 < {} \\\lor n1+n2 < {} = {s1:?}", g-1, C-1);
println!("S2 = (n1, n2) \\\in X: n1+n2 < {} = {s2:?}", C-1);

```

S1 = (n1, n2) \\\in X: n1 < 2 \\\lor n1+n2 < 3 = [(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1)]

S2 = (n1, n2) \\\in X: n1+n2 < 3 = [(0, 0), (0, 1), (0, 2), (0, 3), (1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (3, 0)]

Для неполнодоступной модели Эрланга с двумя типами заявок известно, что любой замкнутый цикл имеет одинаковое произведение интенсивностей переходов в обе стороны, поэтому по критерию Колмогорова существует частичный баланс, и поэтому можно посчитать распределение вероятностей.

Для этого сначала рассчитаем плотность вероятностей в каждой ячейке как пропорцию от p_{00} , а затем, учитывая, что мы имеем дело с распределением вероятностей, -- определим значение p_{00} относительно него.

```

In [13]: use std::collections::HashMap;

fn make_probs(log: bool, rho_1: &f64, rho_2: &f64, states: &Vec<(i32, i32)>)

    let mut probs = HashMap::new();
    for (n1, n2) in states.iter() {
        let n1 = *n1;
        let n2 = *n2;
        if (n1, n2) != (0, 0) {
            let p = (rr(*rho_1 as f64).pow(n1) / factorial(&u(n1 as usize)))
            probs.insert((n1, n2), p);
        }
    }

```

```

    }
    if log {
        println!("probs before normalization: ");
        for prob in probs.iter() {
            println!("{prob:?}");
        }
    }

    probs.insert((0, 0), R::from_i8(1).unwrap());

    let p00 = probs.iter().map(|(_, v)| v).sum::<R>().pow(-1);

    if log {
        println!("p00 = {} = {}", p00, p00.to_f64().unwrap());
    }

    probs.iter_mut().for_each(|(_, v)| *v *= &p00);

    probs

}

let probs = make_probs(true, &rho_1, &rho_2, &states);
println!("probs after normalization: ");
for prob in probs.iter() {
    println!("{prob:?}");
}

```

```

probs before normalization:
((1, 2), Ratio { numer: 1, denom: 4 })
((2, 0), Ratio { numer: 1, denom: 8 })
((0, 2), Ratio { numer: 1, denom: 2 })
((3, 0), Ratio { numer: 1, denom: 48 })
((2, 2), Ratio { numer: 1, denom: 16 })
((3, 1), Ratio { numer: 1, denom: 48 })
((1, 1), Ratio { numer: 1, denom: 2 })
((0, 4), Ratio { numer: 1, denom: 24 })
((0, 3), Ratio { numer: 1, denom: 6 })
((2, 1), Ratio { numer: 1, denom: 8 })
((1, 0), Ratio { numer: 1, denom: 2 })
((0, 1), Ratio { numer: 1, denom: 1 })
((1, 3), Ratio { numer: 1, denom: 12 })
p00 = 48/211 = 0.22748815165876776
probs after normalization:
((1, 2), Ratio { numer: 12, denom: 211 })
((2, 0), Ratio { numer: 6, denom: 211 })
((0, 2), Ratio { numer: 24, denom: 211 })
((3, 0), Ratio { numer: 1, denom: 211 })
((2, 2), Ratio { numer: 3, denom: 211 })
((3, 1), Ratio { numer: 1, denom: 211 })
((1, 1), Ratio { numer: 24, denom: 211 })
((0, 4), Ratio { numer: 2, denom: 211 })
((0, 3), Ratio { numer: 8, denom: 211 })
((2, 1), Ratio { numer: 6, denom: 211 })
((0, 0), Ratio { numer: 48, denom: 211 })
((1, 0), Ratio { numer: 24, denom: 211 })
((0, 1), Ratio { numer: 48, denom: 211 })
((1, 3), Ratio { numer: 4, denom: 211 })

```

Out[13]: ()

Вероятность блокировки запроса на предоставление каждого из типов услуг -- это сумма вероятностей того, что система окажется в одном из блокирующих состояний.

```

In [14]: println!("B_1 = sum(p(n1, n2)), (n1, n2) \in B1 = {}", b1.iter().map(|loc|
println!("B_2 = sum(p(n1, n2)), (n1, n2) \in B2 = {}", b2.iter().map(|loc|

```

```

B_1 = sum(p(n1, n2)), (n1, n2) \in B1 = 11/211
B_2 = sum(p(n1, n2)), (n1, n2) \in B2 = 10/211

```

Среднее число обслуживаемых запросов каждого типа -- это сумма вероятностей оказаться в каждом из состояний, умноженная на количество заявок данного типа, которое обрабатывается в этом состоянии. Для заявок первого типа это первая координата, а для второго типа -- вторая координата.

```

In [15]: fn rrr(n: i64) -> R {
    R::from_i64(n).unwrap()
}
fn avg_serviced_requests(probs: &HashMap<(i32, i32), R>) -> (R, R) {
    let N1 = probs.iter().map(|((n1, n2), p)| rrr(*n1 as i64) * p).sum::<R>()
    let N2 = probs.iter().map(|((n1, n2), p)| rrr(*n2 as i64) * p).sum::<R>()
    (N1, N2)
}

```



```

}

let (N1, N2) = avg_served_requests(&probs);
println!("N1 = {} = {}", N1, N1.to_f64().unwrap());
println!("N2 = {} = {}", N2, N2.to_f64().unwrap());

```

N1 = 100/211 = 0.47393364928909953

N2 = 201/211 = 0.95260663507109

Общее среднее число обрабатываемых заявок равно сумме двух разных средних значений.

```

In [16]: let N = N1 + N2;
println!("N = {} = {}", N, N.to_f64().unwrap());

```

N = 301/211 = 1.4265402843601895

4. графики

График вероятности блокировки заявки можно определить, варьируя значение λ и измеряя вероятность блокировки каждой из заявок. Здесь потребуются четыре графика: два, когда варьируется λ_1 и два для λ_2 ; в обоих случаях надо рассмотреть вероятность блокировки первого и второго типа заявки.

```

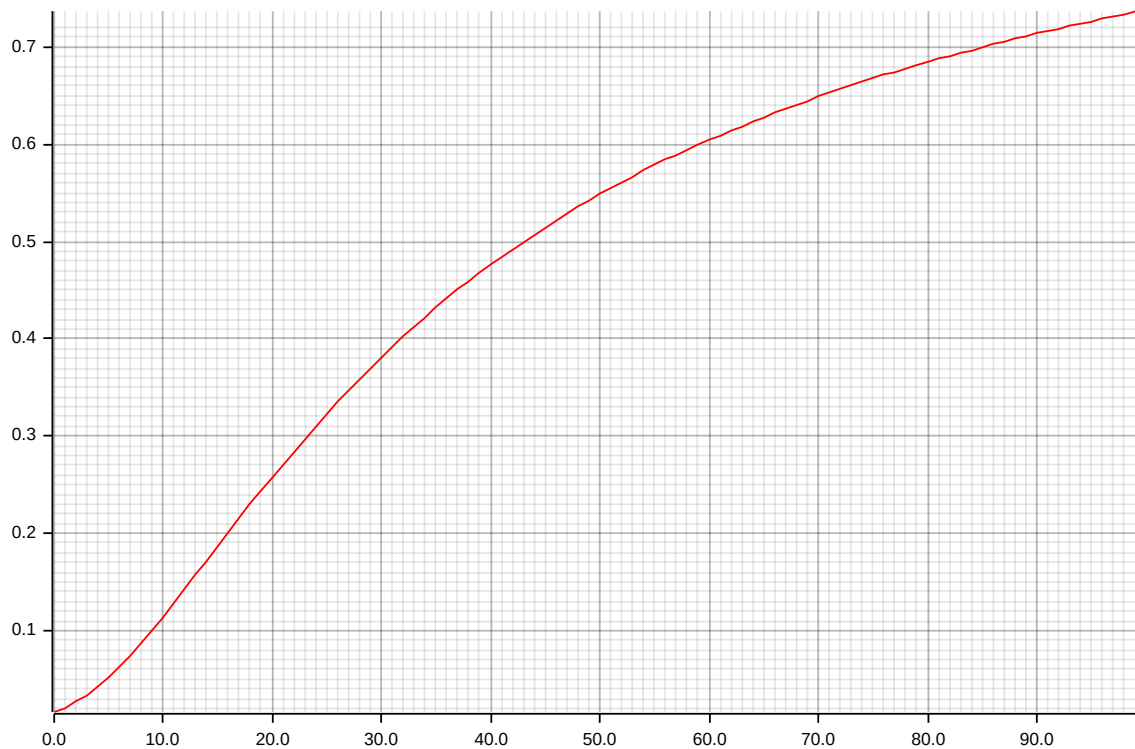
In [18]: let mut blocking_probs1 = vec![];
let mut blocking_probs2 = vec![];
for lambda1 in 0..100 {
    let rho_1 = lambda1 as f64 / &mu1;
    let probs = make_probs(false, &rho_1, &rho_2, &states);
    let blocking_prob1 = probs.iter().filter(|(v,_)| b1.contains(v)).map(|_|
    let blocking_prob2 = probs.iter().filter(|(v,_)| b2.contains(v)).map(|_|
    blocking_probs1.push((lambda1 as f32, blocking_prob1.to_f32().unwrap()))
    blocking_probs2.push((lambda1 as f32, blocking_prob2.to_f32().unwrap()))
}
println!("{blocking_probs1:?}");
draw_chart(&blocking_probs1, "lambda1 vs B_1")

```

```
[(0.0, 0.015384615), (1.0, 0.020471914), (2.0, 0.026611352), (3.0, 0.0339317
7), (4.0, 0.042455584), (5.0, 0.0521327), (6.0, 0.062866606), (7.0, 0.0745339
7), (8.0, 0.08699891), (9.0, 0.10012309), (10.0, 0.11377245), (11.0, 0.127821
53), (12.0, 0.14215587), (13.0, 0.15667325), (14.0, 0.17128387), (15.0, 0.185
90999), (16.0, 0.20048526), (17.0, 0.21495377), (18.0, 0.22926901), (19.0, 0.
24339284), (20.0, 0.25729442), (21.0, 0.27094936), (22.0, 0.2843387), (23.0,
0.2974483), (24.0, 0.3102679), (25.0, 0.3227907), (26.0, 0.33501273), (27.0,
0.34693238), (28.0, 0.35855), (29.0, 0.36986756), (30.0, 0.38088828), (31.0,
0.3916165), (32.0, 0.40205732), (33.0, 0.41221657), (34.0, 0.42210054), (35.
0, 0.43171585), (36.0, 0.44106945), (37.0, 0.45016837), (38.0, 0.45901984),
(39.0, 0.46763104), (40.0, 0.47600913), (41.0, 0.4841613), (42.0, 0.4920945
5), (43.0, 0.4998158), (44.0, 0.50733185), (45.0, 0.51464933), (46.0, 0.52177
465), (47.0, 0.5287141), (48.0, 0.5354739), (49.0, 0.54205984), (50.0, 0.5484
7777), (51.0, 0.5547331), (52.0, 0.5608313), (53.0, 0.5667775), (54.0, 0.5725
7676), (55.0, 0.57823384), (56.0, 0.58375347), (57.0, 0.58914), (58.0, 0.5943
979), (59.0, 0.59953123), (60.0, 0.60454404), (61.0, 0.6094402), (62.0, 0.614
2234), (63.0, 0.6188973), (64.0, 0.62346524), (65.0, 0.6279306), (66.0, 0.632
2965), (67.0, 0.63656604), (68.0, 0.6407423), (69.0, 0.64482796), (70.0, 0.64
88259), (71.0, 0.6527387), (72.0, 0.6565689), (73.0, 0.660319), (74.0, 0.6639
913), (75.0, 0.66758806), (76.0, 0.6711116), (77.0, 0.6745639), (78.0, 0.6779
471), (79.0, 0.68126315), (80.0, 0.68451387), (81.0, 0.68770117), (82.0, 0.69
082683), (83.0, 0.6938925), (84.0, 0.69689983), (85.0, 0.69985044), (86.0, 0.
70274585), (87.0, 0.7055875), (88.0, 0.70837694), (89.0, 0.7111154), (90.0,
0.71380436), (91.0, 0.71644497), (92.0, 0.7190386), (93.0, 0.7215864), (94.0,
0.7240895), (95.0, 0.72654915), (96.0, 0.72896636), (97.0, 0.7313422), (98.0,
0.7336776), (99.0, 0.7359738)]
```

Out[18]:

lambda1 vs B_1

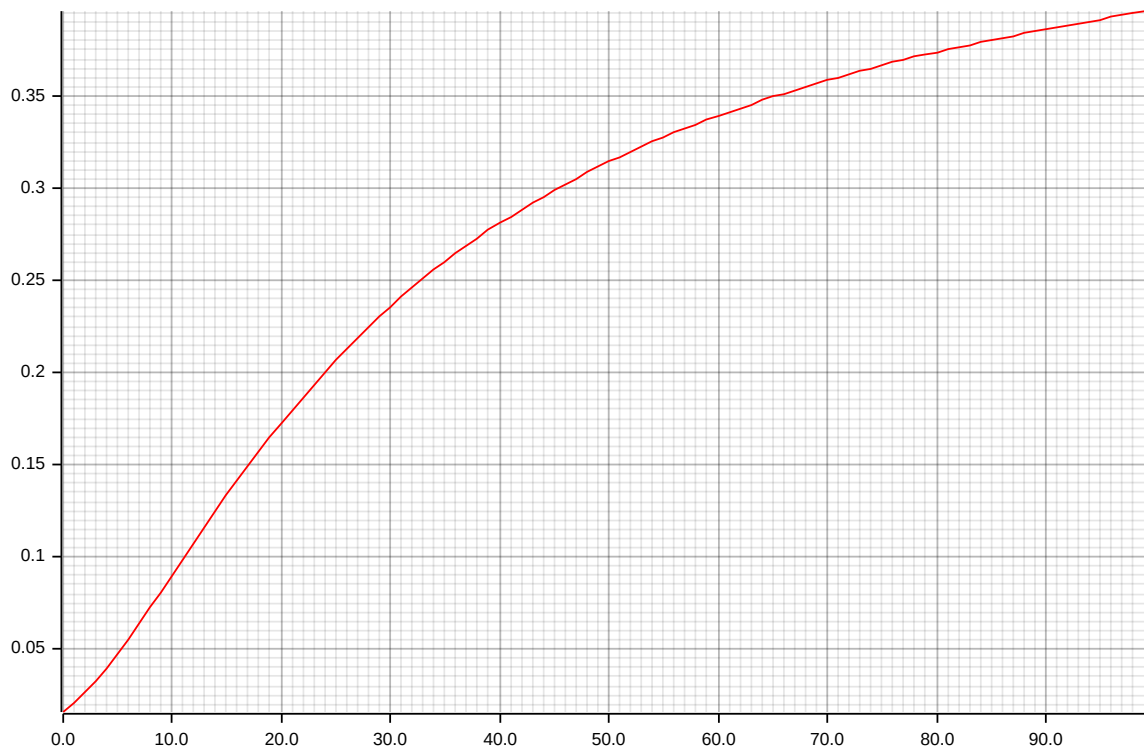


```
In [19]: println!("{blocking_probs2:?}");
draw_chart(&blocking_probs2, "lambda1 vs B_2")
```

```
[(0.0, 0.015384615), (1.0, 0.020416131), (2.0, 0.026206618), (3.0, 0.03269204), (4.0, 0.03978647), (5.0, 0.047393367), (6.0, 0.055413704), (7.0, 0.06375153), (8.0, 0.07231749), (9.0, 0.08103082), (10.0, 0.08982036), (11.0, 0.09862461), (12.0, 0.10739146), (13.0, 0.116077535), (14.0, 0.124647334), (15.0, 0.1330724), (16.0, 0.1413304), (17.0, 0.14940427), (18.0, 0.15728146), (19.0, 0.16495317), (20.0, 0.1724138), (21.0, 0.17966032), (22.0, 0.18669185), (23.0, 0.19350925), (24.0, 0.20011474), (25.0, 0.20651163), (26.0, 0.21270406), (27.0, 0.21869686), (28.0, 0.22449528), (29.0, 0.23010492), (30.0, 0.23553163), (31.0, 0.24078134), (32.0, 0.24586007), (33.0, 0.25077382), (34.0, 0.25552854), (35.0, 0.26013008), (36.0, 0.26458415), (37.0, 0.26889643), (38.0, 0.2730723), (39.0, 0.27711707), (40.0, 0.2810358), (41.0, 0.28483343), (42.0, 0.2885147), (43.0, 0.29208416), (44.0, 0.2955462), (45.0, 0.29890501), (46.0, 0.3021646), (47.0, 0.30532888), (48.0, 0.30840147), (49.0, 0.31138596), (50.0, 0.31428573), (51.0, 0.31710398), (52.0, 0.31984383), (53.0, 0.32250825), (54.0, 0.32510003), (55.0, 0.3276219), (56.0, 0.3300765), (57.0, 0.33246619), (58.0, 0.3347934), (59.0, 0.33706036), (60.0, 0.33926928), (61.0, 0.34142217), (62.0, 0.34352106), (63.0, 0.34556782), (64.0, 0.34756425), (65.0, 0.3495121), (66.0, 0.35141304), (67.0, 0.35326862), (68.0, 0.35508043), (69.0, 0.35684988), (70.0, 0.35857838), (71.0, 0.36026728), (72.0, 0.36191785), (73.0, 0.36353135), (74.0, 0.36510897), (75.0, 0.3666518), (76.0, 0.368161), (77.0, 0.36963755), (78.0, 0.37108248), (79.0, 0.37249678), (80.0, 0.3738814), (81.0, 0.37523717), (82.0, 0.37656498), (83.0, 0.37786567), (84.0, 0.37914005), (85.0, 0.38038886), (86.0, 0.3816128), (87.0, 0.38281268), (88.0, 0.3839891), (89.0, 0.38514277), (90.0, 0.38627428), (91.0, 0.3873843), (92.0, 0.3884734), (93.0, 0.3895421), (94.0, 0.39059103), (95.0, 0.3916207), (96.0, 0.3926316), (97.0, 0.39362422), (98.0, 0.39459905), (99.0, 0.39555657)]
```

Out[19]:

lambda1 vs B_2



```
In [20]: let mut blocking_probs1 = vec![];
let mut blocking_probs2 = vec![];
for lambda2 in 0..100 {
```

```

    let rho_2 = lambda2 as f64 / &mu1;
    let probs = make_probs(false, &rho_1, &rho_2, &states);
    let blocking_prob1 = probs.iter().filter(|(v,_)| b1.contains(v)).map(|_|
    let blocking_prob2 = probs.iter().filter(|(v,_)| b2.contains(v)).map(|_|
    blocking_probs1.push((lambda2 as f32, blocking_prob1.to_f32().unwrap()))
    blocking_probs2.push((lambda2 as f32, blocking_prob2.to_f32().unwrap()))
}
println!("{blocking_probs1:?}");
draw_chart(&blocking_probs1, "lambda2 vs B_1")

```

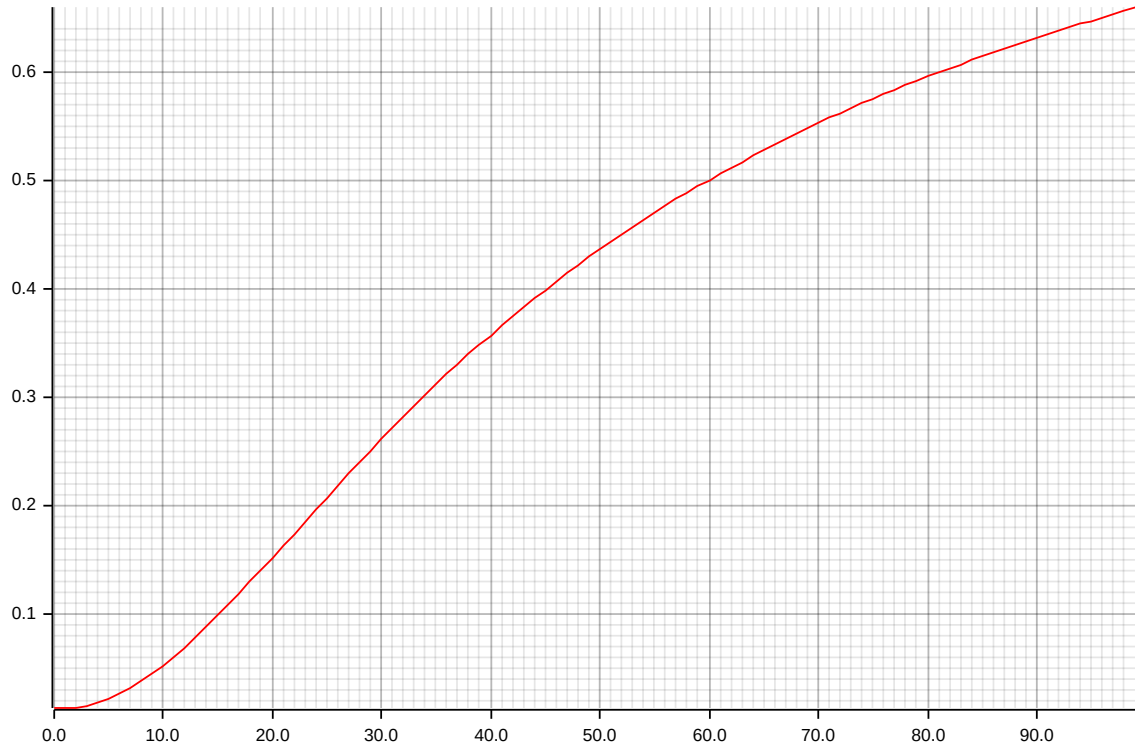
```

[(0.0, 0.012658228), (1.0, 0.012991655), (2.0, 0.014049463), (3.0, 0.01590048
7), (4.0, 0.018589282), (5.0, 0.02213667), (6.0, 0.026541755), (7.0, 0.031784
83), (8.0, 0.037830684), (9.0, 0.044632025), (10.0, 0.0521327), (11.0, 0.0602
7065), (12.0, 0.068980455), (13.0, 0.07819549), (14.0, 0.08784965), (15.0, 0.
09787867), (16.0, 0.10822119), (17.0, 0.11881937), (18.0, 0.12961943), (19.0,
0.14057185), (20.0, 0.15163147), (21.0, 0.16275752), (22.0, 0.17391339), (23.
0, 0.1850665), (24.0, 0.19618814), (25.0, 0.20725307), (26.0, 0.21823932), (2
7.0, 0.22912799), (28.0, 0.23990281), (29.0, 0.25055003), (30.0, 0.26105812),
(31.0, 0.27141747), (32.0, 0.28162032), (33.0, 0.29166043), (34.0, 0.3015329
2), (35.0, 0.31123418), (36.0, 0.3207616), (37.0, 0.33011356), (38.0, 0.33928
925), (39.0, 0.34828848), (40.0, 0.35711178), (41.0, 0.36576012), (42.0, 0.37
42349), (43.0, 0.38253796), (44.0, 0.3906714), (45.0, 0.39863762), (46.0, 0.4
0643916), (47.0, 0.41407883), (48.0, 0.42155954), (49.0, 0.4288843), (50.0,
0.4360562), (51.0, 0.4430784), (52.0, 0.4499541), (53.0, 0.45668653), (54.0,
0.46327886), (55.0, 0.46973437), (56.0, 0.47605622), (57.0, 0.4822476), (58.
0, 0.48831165), (59.0, 0.49425143), (60.0, 0.50007004), (61.0, 0.5057704), (6
2.0, 0.5113555), (63.0, 0.5168283), (64.0, 0.5221915), (65.0, 0.52744794), (6
6.0, 0.5326003), (67.0, 0.5376512), (68.0, 0.5426032), (69.0, 0.5474588), (7
0.0, 0.5522206), (71.0, 0.5568908), (72.0, 0.56147176), (73.0, 0.5659658), (7
4.0, 0.570375), (75.0, 0.57470167), (76.0, 0.5789477), (77.0, 0.5831153), (7
8.0, 0.5872063), (79.0, 0.59122264), (80.0, 0.59516615), (81.0, 0.5990387),
(82.0, 0.602842), (83.0, 0.6065777), (84.0, 0.61024755), (85.0, 0.6138531),
(86.0, 0.61739594), (87.0, 0.6208775), (88.0, 0.62429935), (89.0, 0.6276628
4), (90.0, 0.63096946), (91.0, 0.6342204), (92.0, 0.63741714), (93.0, 0.64056
087), (94.0, 0.6436528), (95.0, 0.6466941), (96.0, 0.6496861), (97.0, 0.65262
973), (98.0, 0.6555262), (99.0, 0.65837663)]

```

Out[20]:

lambda2 vs B_1



```
In [21]: println!("{blocking_probs2:?}");  
draw_chart(&blocking_probs2, "lambda2 vs B_2")
```

```
[(0.0, 0.0), (1.0, 0.0015371892), (2.0, 0.003682386), (3.0, 0.006515239), (4.  
0, 0.010090158), (5.0, 0.014436958), (6.0, 0.01956296), (7.0, 0.025455963),  
(8.0, 0.032087624), (9.0, 0.03941691), (10.0, 0.047393367), (11.0, 0.05596012  
2), (12.0, 0.06505647), (13.0, 0.07462005), (14.0, 0.08458862), (15.0, 0.0949  
01375), (16.0, 0.105500095), (17.0, 0.11632977), (18.0, 0.12733912), (19.0,  
0.13848095), (20.0, 0.14971209), (21.0, 0.16099359), (22.0, 0.1722905), (23.  
0, 0.1835717), (24.0, 0.19480975), (25.0, 0.2059806), (26.0, 0.21706334), (2  
7.0, 0.22803995), (28.0, 0.23889506), (29.0, 0.24961564), (30.0, 0.2601908),  
(31.0, 0.2706116), (32.0, 0.28087077), (33.0, 0.29096252), (34.0, 0.3008824  
6), (35.0, 0.3106273), (36.0, 0.3201949), (37.0, 0.32958385), (38.0, 0.338793  
67), (39.0, 0.34782442), (40.0, 0.35667682), (41.0, 0.36535206), (42.0, 0.373  
8518), (43.0, 0.38217795), (44.0, 0.39033282), (45.0, 0.39831892), (46.0, 0.4  
06139), (47.0, 0.41379586), (48.0, 0.42129257), (49.0, 0.42863223), (50.0, 0.  
43581805), (51.0, 0.44285324), (52.0, 0.44974107), (53.0, 0.45648482), (54.0,  
0.46308777), (55.0, 0.4695532), (56.0, 0.47588438), (57.0, 0.48208445), (58.  
0, 0.48815668), (59.0, 0.49410418), (60.0, 0.49993), (61.0, 0.50563717), (62.  
0, 0.5112287), (63.0, 0.5167075), (64.0, 0.52207637), (65.0, 0.52733815), (6  
6.0, 0.5324955), (67.0, 0.53755116), (68.0, 0.54250765), (69.0, 0.5473676),  
(70.0, 0.5521333), (71.0, 0.55680734), (72.0, 0.5613919), (73.0, 0.5658893),  
(74.0, 0.57030183), (75.0, 0.5746315), (76.0, 0.5788805), (77.0, 0.58305085),  
(78.0, 0.58714443), (79.0, 0.5911633), (80.0, 0.59510916), (81.0, 0.5989839  
4), (82.0, 0.6027894), (83.0, 0.60652715), (84.0, 0.6101989), (85.0, 0.613806  
3), (86.0, 0.6173509), (87.0, 0.6208342), (88.0, 0.6242576), (89.0, 0.6276226  
6), (90.0, 0.6309307), (91.0, 0.6341831), (92.0, 0.63738114), (93.0, 0.640526  
1), (94.0, 0.64361924), (95.0, 0.6466618), (96.0, 0.64965487), (97.0, 0.65259  
96), (98.0, 0.65549713), (99.0, 0.6583485)]
```

Out[21]:

lambda2 vs B_2

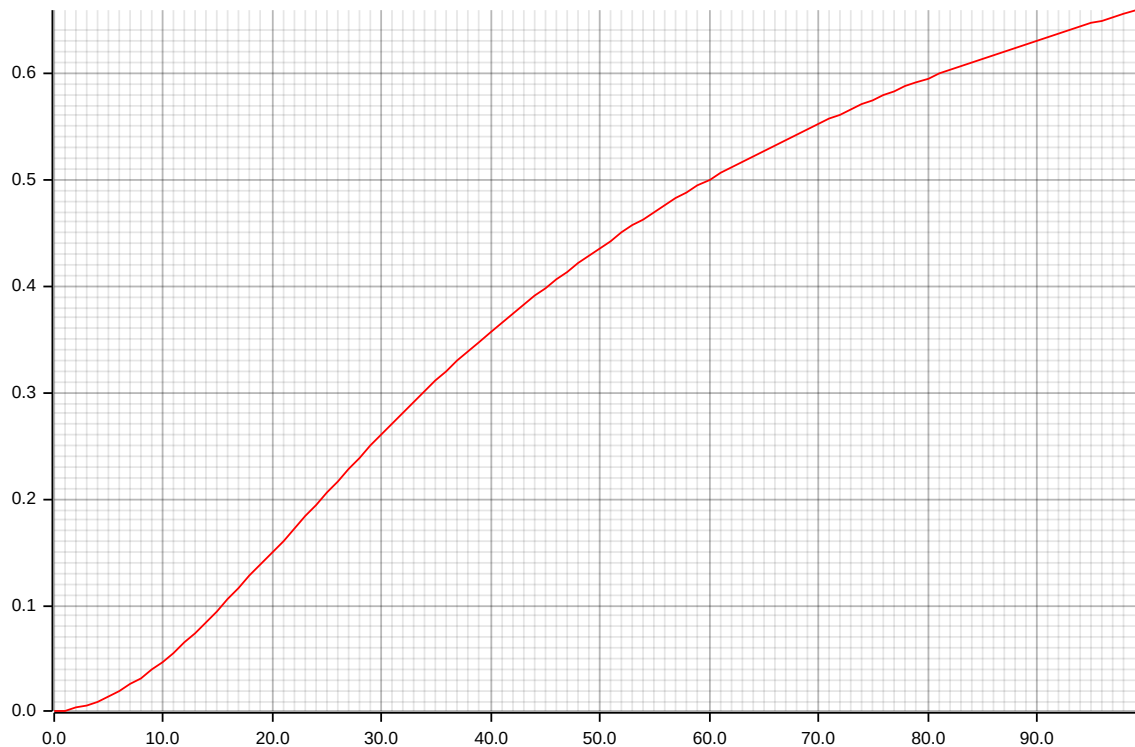


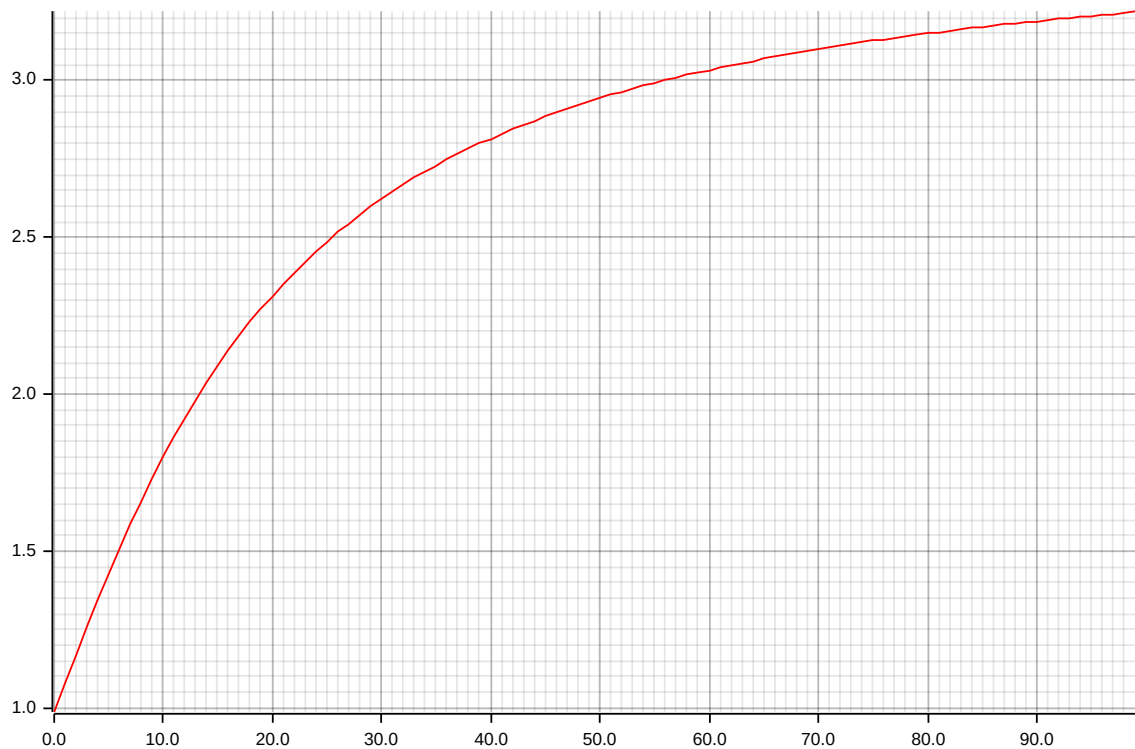
График зависимости среднего числа обслуживаемых заявок можно вычислить похожим образом: варьировать интенсивность поступления заявок и измерять среднее число. Здесь можно обойтись двумя графиками, если суммировать количество обслуживаемых заявок между двумя типами.

```
In [22]: let mut avg_svc = vec![];
for lambda1 in 0..100 {
    let rho_1 = lambda1 as f64 / &mu1;
    let probs = make_probs(false, &rho_1, &rho_2, &states);
    let reqs = avg_serviced_requests(&probs);
    let avg_service = reqs.0 + reqs.1;
    avg_svc.push((lambda1 as f32, avg_service.to_f32().unwrap()));
}
println!("{avg_svc:?}");
draw_chart(&avg_svc, "lambda1 vs Nbar")
```

```
[(0.0, 0.9846154), (1.0, 1.0775367), (2.0, 1.1684711), (3.0, 1.2571285), (4.0, 1.3432313), (5.0, 1.4265403), (6.0, 1.5068663), (7.0, 1.5840747), (8.0, 1.6580834), (9.0, 1.7288584), (10.0, 1.7964072), (11.0, 1.8607717), (12.0, 1.9220215), (13.0, 1.9802473), (14.0, 2.0355554), (15.0, 2.0880625), (16.0, 2.1378932), (17.0, 2.1851742), (18.0, 2.2300344), (19.0, 2.2726004), (20.0, 2.3129973), (21.0, 2.351346), (22.0, 2.387763), (23.0, 2.4223597), (24.0, 2.4552424), (25.0, 2.4865117), (26.0, 2.5162628), (27.0, 2.5445857), (28.0, 2.5715647), (29.0, 2.597279), (30.0, 2.6218035), (31.0, 2.6452074), (32.0, 2.6675565), (33.0, 2.6889114), (34.0, 2.7093296), (35.0, 2.7288644), (36.0, 2.7475657), (37.0, 2.7654805), (38.0, 2.7826524), (39.0, 2.7991219), (40.0, 2.8149276), (41.0, 2.8301053), (42.0, 2.8446882), (43.0, 2.858708), (44.0, 2.8721936), (45.0, 2.885173), (46.0, 2.897672), (47.0, 2.9097147), (48.0, 2.9213238), (49.0, 2.9325206), (50.0, 2.9433255), (51.0, 2.953757), (52.0, 2.9638333), (53.0, 2.9735708), (54.0, 2.9829855), (55.0, 2.992092), (56.0, 3.0009043), (57.0, 3.0094357), (58.0, 3.0176988), (59.0, 3.0257053), (60.0, 3.0334663), (61.0, 3.0409925), (62.0, 3.0482936), (63.0, 3.0553792), (64.0, 3.0622582), (65.0, 3.0689392), (66.0, 3.0754302), (67.0, 3.0817387), (68.0, 3.087872), (69.0, 3.093837), (70.0, 3.0996404), (71.0, 3.105288), (72.0, 3.1107862), (73.0, 3.1161404), (74.0, 3.1213558), (75.0, 3.126438), (76.0, 3.131391), (77.0, 3.1362205), (78.0, 3.1409302), (79.0, 3.1455245), (80.0, 3.1500075), (81.0, 3.1543832), (82.0, 3.158655), (83.0, 3.1628265), (84.0, 3.1669014), (85.0, 3.1708825), (86.0, 3.174773), (87.0, 3.1785758), (88.0, 3.182294), (89.0, 3.18593), (90.0, 3.1894867), (91.0, 3.1929665), (92.0, 3.1963716), (93.0, 3.1997044), (94.0, 3.2029674), (95.0, 3.2061625), (96.0, 3.2092915), (97.0, 3.2123568), (98.0, 3.21536), (99.0, 3.218303)]
```

Out[22]:

lambda1 vs Nbar



```
In [25]: let mut avg_svc = vec![];
for lambda2 in 0..100 {
    let rho_2 = lambda2 as f64 / &mu1;
    let probs = make_probs(false, &rho_1, &rho_2, &states);
```

```

    let reqs = avg_serviced_requests(&probs);
    let avg_service = reqs.0 + reqs.1;
    avg_svc.push((lambda2 as f32, avg_service.to_f32().unwrap()));
}
println!("{avg_svc:?}");
draw_chart(&avg_svc, "lambda2 vs Nbar")

```

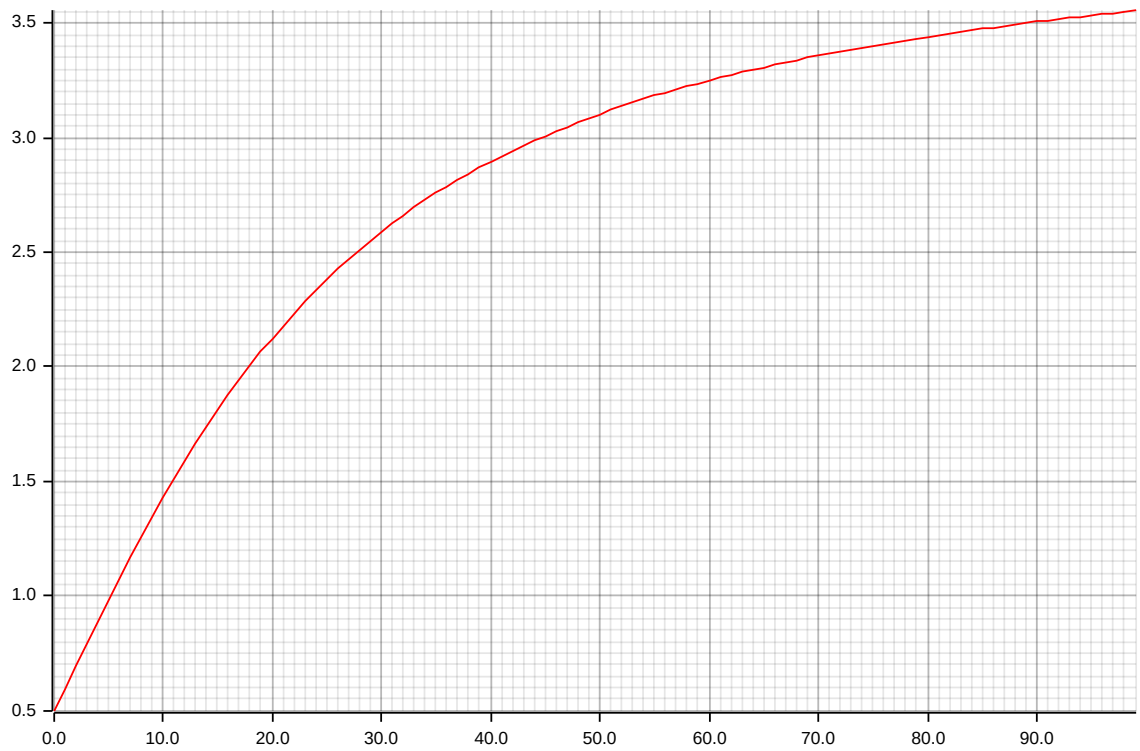
```

[(0.0, 0.49367088), (1.0, 0.59335047), (2.0, 0.6922388), (3.0, 0.7900952),
(4.0, 0.8866693), (5.0, 0.9817132), (6.0, 1.0749913), (7.0, 1.1662884), (8.0,
1.2554146), (9.0, 1.3422087), (10.0, 1.4265403), (11.0, 1.5083085), (12.0, 1.
587442), (13.0, 1.6638962), (14.0, 1.7376511), (15.0, 1.8087085), (16.0, 1.87
70893), (17.0, 1.9428297), (18.0, 2.0059798), (19.0, 2.0666003), (20.0, 2.124
7602), (21.0, 2.1805346), (22.0, 2.2340043), (23.0, 2.2852519), (24.0, 2.3343
625), (25.0, 2.381422), (26.0, 2.4265156), (27.0, 2.4697282), (28.0, 2.511142
5), (29.0, 2.5508397), (30.0, 2.5888984), (31.0, 2.6253953), (32.0, 2.660403
5), (33.0, 2.6939936), (34.0, 2.7262332), (35.0, 2.7571874), (36.0, 2.786917
4), (37.0, 2.8154829), (38.0, 2.8429394), (39.0, 2.8693407), (40.0, 2.894736
8), (41.0, 2.9191763), (42.0, 2.942705), (43.0, 2.965366), (44.0, 2.9871998),
(45.0, 3.008246), (46.0, 3.028541), (47.0, 3.04812), (48.0, 3.067016), (49.0,
3.08526), (50.0, 3.1028817), (51.0, 3.1199093), (52.0, 3.1363695), (53.0, 3.1
522872), (54.0, 3.1676867), (55.0, 3.1825902), (56.0, 3.1970193), (57.0, 3.21
09947), (58.0, 3.2245355), (59.0, 3.2376597), (60.0, 3.250385), (61.0, 3.2627
28), (62.0, 3.2747045), (63.0, 3.2863288), (64.0, 3.2976155), (65.0, 3.308578
3), (66.0, 3.3192296), (67.0, 3.3295817), (68.0, 3.3396463), (69.0, 3.349434
4), (70.0, 3.3589566), (71.0, 3.3682227), (72.0, 3.3772426), (73.0, 3.38602
5), (74.0, 3.394579), (75.0, 3.4029129), (76.0, 3.4110343), (77.0, 3.418951),
(78.0, 3.4266703), (79.0, 3.4341989), (80.0, 3.4415436), (81.0, 3.4487107),
(82.0, 3.4557061), (83.0, 3.4625359), (84.0, 3.4692051), (85.0, 3.4757197),
(86.0, 3.4820843), (87.0, 3.488304), (88.0, 3.4943833), (89.0, 3.5003269), (9
0.0, 3.5061388), (91.0, 3.5118237), (92.0, 3.517385), (93.0, 3.5228267), (94.
0, 3.5281525), (95.0, 3.533366), (96.0, 3.5384703), (97.0, 3.543469), (98.0,
3.548365), (99.0, 3.5531616)]

```


Out[25]:

lambda2 vs Nbar



In []: